VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Ad-hoc komunikační síť pro Android

Ad-hoc Network for Android

2015/2016

Bc. Martin Lapiš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Martin Lapiš**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Ad-hoc komunikační síť pro android Ad-hoc Network for Android
Jazyk vypracování:	čeština

Zásady pro vypracování:

Cílem této diplomové práce je vytvořit aplikaci, která umožní zabezpečenou komunikaci mezi klienty v síti navzájem připojených mobilů pomocí ad-hoc sítí.

1. Vytvořte aplikaci, která vyhledá pomocí wifi a bluetooth běžící aplikace ve svém okolí a připojí se k nim do sítě. Síť vytvořená aplikacemi může být veřejná, nebo zabezpečená a umožnit připojení jen po autorizaci.
2. Každá aplikace bude fungovat jako uzel pro ostatní aplikace. Síť bude schopná se vyrovnat s neočekávanými výpadky jednotlivých uzlů. Navrhněte a implementujte protokol zajišťující tyto požadavky.
3. Vyzkoušejte síť na reálné aplikaci, která bude umět zasílání zpráv vybraným klientům, zprávy budou šifrovány.

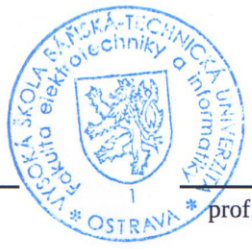
Seznam doporučené odborné literatury:

[1] Bill Phillips, Brian Hardy: Android Programming: The Big Nerd Ranch Guide, Big Nerd Ranch Guides; 1 edition (April 7, 2013)
[2] Reto Meier: Professional Android 4 Application Development, Wrox; 3 edition (May 1, 2012)
Další literatura podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Svatopluk Štolfa, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 29.04.2016



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: 27. dubna 2016

podpis studenta

Poděkování

Rád bych poděkoval panu Ing. Svatopluku Štolfovi, Ph.D., za odbornou pomoc a konzultaci při vytváření této diplomové práce.

Abstrakt

Tato diplomová práce je souhrnným popisem Ad-hoc komunikace na platformě operačního systému Android. Zároveň tato práce popisuje proces vytvoření ukázkové aplikace, která umožňuje zabezpečenou a nezabezpečenou textovou komunikaci pomocí v Androidu podporovaných technologii Bluetooth a Wi-Fi Direct.

Android pomocí těchto technologií neumožňuje plně využít Ad-hoc způsobu komunikace. Technologie Bluetooth a Wi-Fi Direct obecně umožňují komunikaci pouze ve skupině a jen mezi klientem a serverem. Zároveň neobsahují žádnou možnost komunikace mezi klienty. Pro obě technologie na platformě Android existují pouze základní softwarové nástroje, které lze využít a které obsahují některá omezení. Tyto softwarové nástroje pokrývají základní funkce obou technologií a také neobsahují žádnou implementaci komunikace mezi klienty. Komunikace mezi klienty se musí vždy řešit na úrovni aplikace.

Výsledná aplikace pro operační systém Android obsahuje oddělenou textovou komunikaci pomocí implementace softwarových nástrojů pro Bluetooth a Wi-Fi Direct a základní možnost přeposílaní zpráv mezi klienty. Pro vývoj aplikace je využito programu Android Studio společně s poslední verzí vývojářských nástrojů.

Klíčová slova

Síť, Android, Java, Bluetooth, Bluetooth LE, Wi-Fi, Wi-Fi Direct, Ad-hoc, Piconet, Scatternet

Abstract

This diploma thesis is a summary description of the Ad-hoc communication on the platform of operating system Android. Also this thesis describes the process of creating a sample application that enables secure and insecure text communication using in Android supported technologies Bluetooth and Wi-Fi Direct.

Android using these technologies does not fully utilize Ad-hoc communication method. Bluetooth and Wi-Fi Direct generally allow communication only in groups and only between client and server. They contain no possibility of communication between clients. For both technologies on the Android platform, there are only the basic software tools that can be used and which contain some restrictions. These software tools cover the basic features of both technologies and also contain no implementation of communication between clients. Communication between clients must always be solved at the application level.

The final application for Android operating system contains a separate text communication using implementation of software tools for Bluetooth and Wi-Fi Direct and basic possibilities of sending messages between clients. For application development is used a program Android Studio together with the latest version of developer tools.

Key words

Network, Android, Java, Bluetooth, Bluetooth LE, Wi-Fi, Wi-Fi Direct, Ad-hoc, Piconet, Scatternet

Obsah

Seznam použitých symbolů ..... - 10 -
Seznam použitých zkratek ..... - 11 -
Seznam ilustrací a seznam tabulek ..... - 13 -
Úvod ..... - 14 -
1 Android ..... - 15 -
1.1 Verze ..... - 15 -
1.2 Vývoj aplikací ..... - 16 -
1.2.1 Podpůrné knihovny ..... - 17 -
1.3 Základní části aplikace ..... - 18 -
1.3.1 Activity ..... - 18 -
1.3.2 Fragment ..... - 18 -
1.3.3 Service ..... - 19 -
1.3.4 AIDL ..... - 20 -
1.3.5 Application ..... - 20 -
1.3.6 Android Manifest ..... - 20 -
1.3.7 Broadcast Receiver ..... - 21 -
1.3.8 Context ..... - 21 -
1.3.9 Povolení ..... - 21 -
2 Ad-hoc ..... - 22 -
2.1 Ad-hoc v Androidu ..... - 22 -
2.1.1 Ad-hoc přes Bluetooth ..... - 22 -
2.1.2 Ad-hoc přes Wi-Fi Direct ..... - 24 -
3 Bluetooth ..... - 26 -
3.1 Verze ..... - 26 -
3.1.1 Označení verzí ..... - 27 -
3.2 Bluetooth v Androidu ..... - 28 -
3.2.1 Stack ..... - 28 -
3.2.2 Android API v aplikaci ..... - 30 -
3.2.3 Nalezení a párování zařízení ..... - 30 -
3.2.4 Připojení jako klient a zabezpečení ..... - 33 -
3.2.5 Tvorba serveru a komunikace ..... - 34 -

3.2.6 Povolení - 35 -
4 Wi-Fi Direct - 36 -
4.1 Wi-Fi Direct v Androidu - 36 -
4.1.1 Android API v aplikaci - 37 -
4.1.2 Nalezení zařízení nebo služby - 38 -
4.1.3 Vytvoření a detekce služby - 39 -
4.1.4 Párování, připojení k zařízení a zabezpečení - 41 -
4.1.5 Vytvoření skupiny a komunikace - 41 -
4.1.6 Informace o skupině a spojení - 42 -
4.1.7 Povolení - 43 -
5 Aplikace - 44 -
5.1 Architektura - 44 -
5.2 Části aplikace - 45 -
5.2.1 Část Base - 45 -
5.2.2 Část Statistics - 46 -
5.2.3 Část Other - 46 -
5.2.4 Část Adapter - 46 -
5.2.5 Část Widget - 46 -
5.2.6 Část Utils - 47 -
5.2.7 Část Database - 47 -
5.3 Část Bluetooth - 48 -
5.3.1 Klient a server - 48 -
5.3.2 Services - 50 -
5.3.3 Komunikace mezi klienty - 50 -
5.3.4 Notifikace a Broadcast Receivery - 51 -
5.4 Část Wifi - 53 -
5.4.1 Klient a vlastník skupiny - 53 -
5.4.2 Services - 55 -
5.4.3 Komunikace mezi klienty - 55 -
5.4.4 Notifikace a Broadcast Receivery - 56 -
5.5 Uživatelské rozhraní a vzhled - 58 -
5.5.1 Využité knihovny - 58 -
5.5.2 Další knihovny - 59 -

Závěr ..... - 61 -
Použitá literatura ..... - 62 -
Seznam příloh ..... - 64 -

Seznam použitých symbolů

Symbol	Jednotky	Význam symbolu
f	GHz	Frekvence

Seznam použitých zkratek

Zkratka	Význam
ADT	Vývojářské nástroje pro Android
AES	Šifrovací standard
AIDL	Jazyk pro definici rozhraní v Androidu
AODV	Ad hoc vektorový vzdálenostní protokol na vyžádání
AP	Přístupový bod
API	Souhrn nástrojů pro vývoj aplikací
BT_ADDR	Adresa Bluetooth zařízení
DSDV	Cílový vektorový sekvenční vzdálenostní protokol
E-R diagram	Diagram pro zobrazení struktury databáze
ID	Jedinečné identifikační číslo
IPC	Mezi procesová komunikace
IPv6	Síťový protokol
LE	Nízkoenergetická varianta Bluetooth
LTE	Technologie vysokorychlostního internetu v mobilních sítích
MAC	Adresa síťového zařízení
NDK	Nativní vývojářský kit
NSD	Vyhledávání Wi-Fi služby v Androidu
OLSR	Optimalizovaný odkazový stavový směrovací protokol
OS	Operační systém
P2P	Přímá komunikace mezi dvěma klienty
PIN	Osobní identifikační číslo
RFCOMM	Radiofrekvenční komunikační protokol
SDK	Softwarový vývojářský kit
SDP	Bluetooth internetový protokol
TCP	Internetový síťový protokol
UPnP	Sada síťových protokolů
UUID	Univerzální unikátní identifikátor
WPA2	Typ zabezpečení bezdrátové sítě Wi-Fi

WPS	Chráněné nastavení Wi-Fi
XML	Rozšířitelný značkovací jazyk

Seznam ilustrací a seznam tabulek

Číslo ilustrace	Název ilustrace	Číslo stránky
1.1	Logo operačního systému Android	15
1.1	Tabulka verzí OS Android	16
1.2	Schéma sítě Piconet	23
1.3	Schéma sítě Scatternet	24
1.4	Schéma sítě Wi-Fi Direct	25
1.5	Logo Bluetooth	26
1.6	Loga Bluetooth zařízení	27
1.7	Architektura Bluetooth v Androidu	29
1.8	Povolení zviditelnění Bluetooth zařízení	31
1.9	Logo Wi-Fi Direct	36
1.10	Architektura aplikace	45
1.11	Schéma databáze	47
1.12	BtActivity a BtChatClientActivity	49
1.13	BtCreateActivity a BtChatServerActivity	49
1.14	WifiActivity a WifiChatClientActivity	54
1.15	WifiCreateActivity a WifiChatGroupOwnerActivity	54
1.16	Navigační menu aplikace	58
1.17	Grafy ve StatisticsActivity	60
1.18	Knihovna SearchView	60

Úvod

Tato diplomové práce má za cíl shrnout informace k Ad-hoc komunikaci na platformě operačního systému Android. Dále tato práce také popisuje základní proces tvorby Ad-hoc komunikace v implementované aplikaci a také architekturu a funkce této aplikace.

Operační systém Android neumožňuje plné využití Ad-hoc komunikace a tvorbu komunikačního uzlu pro ostatní zařízení. Podporované komunikační technologie Bluetooth a Wi-Fi Direct podporují pouze komunikaci ve skupině a to pouze mezi klientem a serverem. U Bluetooth je základní komunikace možná pouze v rámci sítě Piconet, kterou tvoří skupina o jednom serveru a maximálně sedmi klienty. U Wi-Fi Direct je tato komunikace ve skupině také omezena na jeden server, který se nazývá vlastník skupiny, a jednoho nebo více klientů. Obě technologie neumožňují vzájemnou komunikaci mezi klienty. Tuto komunikaci je nutné zajistit na úrovni aplikace, kde musí server veškerou komunikaci přeposílat.

Implementovaná aplikace umožňuje zabezpečenou a nezabezpečenou síťovou komunikaci navzájem připojených mobilních zařízení pomocí komunikačních technologií Bluetooth a Wi-Fi Direct. Tato aplikace dokáže vyhledat pomocí obou technologií okolní zařízení. Následně se k nim může připojit a lze posílat jednoduché textové zprávy šifrované AES šifrováním. Dále umožňuje tvorbu Bluetooth a Wi-Fi Direct serveru. Aplikace rovněž obsahuje základní implementaci komunikace a přeposílaní zpráv mezi klienty přes server.

Úvodní první kapitola obsahuje popis operačního systému Android. V této kapitole popisuji verze tohoto operačního systému a jejich odlišností. Z důvodů odkazů v následujících kapitolách zde také popisuji, jaké jsou základní součásti aplikace a rovněž způsob vývoje a možností vývoje aplikací pro operační systém Android. Následující druhá kapitola se zaměřuje na Ad-hoc způsob komunikace na platformě operačního systému Android. V této kapitole se blíže zaměřuji na obě podporované technologie síťové komunikace Bluetooth a Wi-Fi Direct. Popisuji zde jejich způsob komunikace, uspořádaní a omezení. Třetí kapitola se věnuje detailně technologii Bluetooth. V této kapitole se nejdříve věnuji obecnému popisu technologie Bluetooth, odlišnostem a popisu jednotlivých verzí. Následně pak popisuji implementaci klasické Bluetooth technologie v operačním systému Android a také základní softwarové nástroje z Android SDK pro klasické Bluetooth. Tyto nástroje byly využity při vývoji Bluetooth části aplikace pro tuto diplomovou práci a všechny ukázkové kódy v této kapitole, jsou vyjmuty přímo z vyvinuté aplikace. Obsahem čtvrté kapitoly je detailní popis Wi-Fi Direct. Kapitola začíná popisem této technologie a pokračuje popisem její implementace v operačním systému Android. Následně pak v kapitole pokračuji popisem softwarových nástrojů z Android SDK pro práci s P2P službou nad Wi-Fi Direct. Tyto softwarové nástroje byly použity pro vývoj Wi-Fi Direct části aplikace pro tuto diplomovou práci. Všechny ukázkové kódy, které obsahuje tato kapitola, jsou vyjmuty přímo z implementované aplikace. Poslední pátá kapitola pojednává o implementaci aplikace pro operační systém Android, která využívá klasické Bluetooth a Wi-Fi Direct P2P službu pro komunikaci. V této kapitole je popsána každá část aplikace. V popisu každé části je uvedeno, jaké obsahuje třídy a jaký je její účel. Části, které se věnují klasickému Bluetooth a Wi-Fi Direct P2P službě obsahují popis komunikačního protokolu, který jim umožňuje komunikovat mezi klienty. Tato kapitola rovněž obsahuje schéma architektury programu, schéma implementované databáze a ukázky uživatelského rozhraní.

1 Android

Obrázek 1.1: *Logo operačního systému Android*

Android je mobilní operační systém založený na jádře Linuxu, který je dostupný jako otevřený software. Každý tak může nalézt na internetu jeho zdrojové kódy a vytvořit si svou vlastní verzi tohoto operačního systému.

První verze Androidu [1] obsahovaly verze jádra Linuxu 2.6. Pozdější verze Androidu využívají novější linuxová jádra. Android využívá mnoho vlastností operačního systému Linux. Využívá například podporu správy paměti, správu sítí, zabudované ovladače, správu procesů a souběžný běh aplikací, které běží jako samostatné procesy s oprávněním stanoveným systémem. Důvodem použití jádra Linux je možnost poměrně snadného sestavení na různých zařízeních. Android tak lze nalézt v zařízeních, jako jsou mobilní telefony nebo tablety, ale i jako varianta Wear [3] v chytrých hodinkách, v automobilech jako varianta Auto nebo dokonce varianta TV [2] v chytrých televizích. Samotná platforma Android nabízí nejen samotný operační systém, ale i kompletní řešení jeho nasazení na mobilní telefony a ostatní zařízení. Rovněž poskytuje pro vývojáře aplikací nástroje pro tvorbu aplikací ve formě programu Android Studio. Pro následné publikování a prodej aplikací existuje obchod Google Play [19], kde uživatelé mohou nalézt velké množství aplikací a her pro všechny verze Androidu.

1.1 Verze

Od první verze 1.0, která byla uvedena v roce 2009, bylo vydáno několik aktualizací a nových verzí, které opravují chyby a přidávají nové funkce. Jednotlivé verze systému jsou anglicky pojmenovány podle sladkostí a liší se verzí API. Aktuální a poslední verzí k dubnu roku 2016 je verze Androidu N s API 23 N. Pro účely této diplomové práce zde uvádím stručnou tabulku všech verzí Androidu společné s jejich jménem, verzí API a bližšími informacemi, které se týkají této diplomové práce. Na tyto informace se budu následně několikrát odkazovat v dalším textu.

Tabulka 1.1: *Tabulka verzí OS Android*

Verze	Jméno	API	Informace k DP
Android 1.0	-	1	
Android 1.1	-	2	
Android 1.5	Cupcake	3	
Android 1.6	Donut	4	
Android 2.0	Eclair	5, 6	Podpora klasického Bluetooth od API 5
Android 2.1	Eclair	7	
Android 2.2	Froyo	8	
Android 2.3	Gingerbread	9, 10	
Android 3.0	Honeycomb	11	
Android 3.1	Honeycomb	12	
Android 3.2	Honeycomb	13	
Android 4.0	Ice Cream Sandwich	14, 15	Podpora Wi-Fi Direct P2P od API 14
Android 4.1	Jelly Bean	16	Podpora Wi-Fi Direct P2P služby
Android 4.2	Jelly Bean	17	Výměna Bluetooth stacku
Android 4.3	Jelly Bean	18	Částečná podpora Bluetooth LE
Android 4.4	KitKat	19	
Android 4.4W	KitKat	20	Podpora Android Wear
Android 5.0	Lollipop	21	Plná podpora Bluetooth LE
Android 5.1	Lollipop	22	
Android 6.0	Marshmallow	23	
Android 7.0	N	24	Vydání asi podzim 2016

1.2 Vývoj aplikací

Android podporuje mnoho vývojových prostředí. Hlavním a jediným oficiálně podporovaným prostředím je v dnešní době Android Studio [11]. Android Studio je vývojové prostředí založené na programu IntelliJ IDEA od společnosti JetBrains. Je speciálně vytvořeno pro vývoj Android aplikací. Je k dispozici ke stažení pro operační systémy Windows, Mac OS X a Linux. Android Studio umožňuje stažení Android SDK a podpůrných knihoven pro všechny verze Androidu. V Android Studiu lze vytvořit aplikace, které budou přesně kopírovat zvyklosti platformy Android. Umožnuje jak programování aplikací, tak i tvorbu rozhraní a obsahuje i velmi omezenou podporu tvorby grafiky pro aplikace.

Jako další vývojové prostředí lze využít Eclipse s nainstalovaným ADT pluginem [12]. Tento plugin již však není Googlem podporován a je doporučováno využití Android Studia. Dále lze využít Xamarin [9] pro Microsoft Visual Studio 2015 s nainstalovanými CrossPlatform Tools a Visual Studio Android Emulator. Při použití obou výše uvedených vývojových prostředí však lze narazit na určité problémy a omezení při vývoji.

Hlavním programovacím jazykem pro tvorbu aplikací je, pro kód aplikace, jazyk Java a pro tvorbu rozhraní a layoutů, jazyk XML. V Android SDK jsou obsaženy základní knihovny

programovacího jazyka Java. Knihovny se svým obsahem blíží platformě Java Standard Edition od společnosti Oracle, ovšem s nepřítomností knihoven pro uživatelské rozhraní z Javy (Abstract Window Toolkit a Swing). Tyto knihovny jsou nahrazeny velkým množstvím knihoven s uživatelským rozhraním pro Android.

Po soudním sporu se společností Oracle, týkající se používání Javy, Google oficiálně přejde v nadcházejícím Android N na otevřenou implementaci Javy OpenJDK. Jde sice stále o kód vycházející z Javy od společnosti Oracle, avšak OpenJDK je otevřený software. Plánuje se tedy přesun knihoven jazyka Java na OpenJDK s cílem vytvoření společného základního kódu pro vývojáře vytvářející aplikace a služby. Tato změna by měla zjednodušit vývoj pro Android N, přičemž vnější změny budou zanedbatelné.

Avšak Java není jediným jazykem pro vývoj aplikací pro Android. Lze využít Microsoft Xamarin [9], ve kterém lze psát aplikace v jazyce C#. Výsledná aplikace je pak postupně překládána nativními kompilátory pro Android, iOS neb Windows. Rovněž lze využít jazyků C a C++ pod Android NDK [10] pro vývoj nativních knihoven a tvorbu rychlejšího kódu. Android NDK umožňuje implementovat v aplikaci nativní kód v C nebo C++. Typickým případem použití jsou různé náročné aplikace jako zdrojové kódy her nebo fyzikální simulace. Tedy použití je velmi specifické a vyplatí se jen v některých případech a jen u některých aplikací a her, jelikož zvyšuje náročnost programování a složitost aplikace. Podpora Android NDK je v aktuálním Android Studiu 2.1 omezená a je v přípravě nová verze NDK nástrojů.

Dále lze využít naprostou novinku vydanou 15. února 2016 a tou je programovací jazyk Kotlin [4]. Kotlin lze zkompilovat i na JavaScript. Vývojářem tohoto jazyku jsou autoři Android Studia JetBrains. Kotlin je navržen tak, aby spolupracoval s kódem v jazyce Java a je závislý na kódu v jazyce Java ze stávající knihovny tříd Java. Pro spolupráci s Android Studiem je do něj nutné doinstalovat plugin.

1.2.1 **Podpůrné knihovny**

Mobilních telefonů je na světě veliké množství, existují v různých verzích a mají nainstalovanou různou verzi Androidu od nejstarší verze 1.0 až po tu nejnovější 6.0.1. Pokud se vývojář rozhodne, že bude jeho program podporovat i starší verze Androidu než je ta aktuální nejnovější, vznikne mu při vývoji řada problému, jelikož starší verze Androidu neumí a nepodporují funkce novějších verzi.

Z tohoto důvodu Google vydává podpůrné knihovny, které tuto situaci s velkým úspěchem dokáží řešit a usnadňují vývojáři práci tím, že využívá jedinou verzi API pro všechny verze Androidu, které bude podporovat jeho program. Balík podpůrných knihoven je souhrn různých tříd a knihoven, které poskytují API, které obsahuje funkce nových verzí Androidu a je zpětně kompatibilní. Rovněž obsahuje některé součásti, které jsou dostupné jedině skrze tyto knihovny. Každá tato knihovna je zpětné kompatibilní do konkrétní verze API.

Nutné je ovšem uvést, že každá podpůrná knihovna není zpětné kompatibilní až k první verzi Androidu 1.0, ale jen k některé novější verzi. Například velmi hodně využívaná knihovna AppCompat je zpětné kompatibilní s API 7, tedy s Androidem od verze 2.1 a výše. Kódy knihoven jsou volně dostupné na internetu. Vývoj aplikace s pomocí těchto podpůrných knihoven je doporučováno jako nejlepší volba, jelikož dochází k zjednodušení programování pro různé verze Androidu. Použití těchto knihoven poskytuje možnost vylepšit vzhled aplikace, zvýšit její výkon a hlavně rozšířit aplikace k více uživatelům z důvodu zvýšené kompatibility aplikace. Aktuální verze těchto knihoven je k dubnu 2016 verze 23.3.0, která je rovněž využita v této diplomové práci.

1.3 Základní části aplikace

V této kapitole rozeberu některé pojmy a části aplikace tak, aby v dalších kapitolách byl vysvětlen jejich význam při implementaci aplikace.

1.3.1 Activity

Třída Activity je důležitou součástí celé aplikace. Jednotlivé instance třídy Activity jsou spouštěny a zpracovávány pod určitým životním cyklem, který zpracovává platforma Android. Třída Activity odpovídá jedné obrazovce, kterou vidí aktuálně uživatel a s kterou může pracovat. Obsahuje tedy grafické rozhraní pro interakci s uživatelem.

Každá Activity má svůj životní cyklus se čtyřmi základními stavy. Pokud je vytvořena nová Activity a je zobrazena na obrazovce, stav Activity je běžící. Předchozí Activity je zastavena a je uložena na pozadí do zásobníku. Uživatel ji neuvidí, dokud běžící Activity není ukončena. Pokud Activity je viditelná, ale uživatel pracuje s jinou Activity, je pozastavená. Zastavené nebo pozastavené aktivity mohou být ukončené samotným systémem, pokud je to nutné třeba v důsledku nedostatku paměti. Pro opětovné zobrazení musí byt tato Activity znovu vytvořena.

Aplikace obsahuje obvykle více Activit, mezi kterými je uživatel schopen přepínat a přitom si Activity může předávat informace pomocí třídy Intent. Samotná třída Activity je potomkem třídy FragmentActivity. Verze Activity, která pracuje s podpůrnými knihovnami a je také potomkem třídy FragmentActivity, se nazývá AppCompatActivity a je zpětně kompatibilní do verze API 7.

1.3.2 Fragment

Od Androidu 3.0 lze v Activity využít třídu Fragment. Fragment je vlastně takovou částí uživatelského rozhraní v Activity. V každé Activity lze využít jeden nebo více Fragmentů a použít je pro vytvoření odlišných funkcí v každém části uživatelského rozhraní. Například Fragment lze využít jako obsah jednotlivých záložek a tabulek. Rovněž lze využít Fragment opakovaně

v různých Activity. Každý Fragment má svůj vlastní životní cyklus, získává svůj vlastní vstup a lze jej přidat a odstranit za běhu Activity. Fragment však musí byt vždy vložen v Activity a jeho životní cyklus je ovlivněn přímo Activity samotnou. Podpůrné knihovny obsahují svou vlastní implementaci Fragmentu pod názvem v4.Fragment. Tato verze je zpětně kompatibilní až do API 4.

1.3.3 **Service**

Třída Service je základní třídou pro všechny Services. Je určena k provádění operací, pro které je nutné běžet na pozadí. Neposkytuje uživatelské rozhraní. Aplikace může spustit Service a ta bude nadále běžet na pozadí, i když se uživatel přepne do jiné aplikace. Tuto třídu lze využít například pro síťové služby, přehrávání hudby, pro provádění náročných operací atd.

Service běží ve stejném procesu jako aplikace, která ji obsahuje a v jejím hlavním vlákně. Nevytváří si vlastní vlákno a neběží ve vlastním procesu. To znamená, že pokud Service pracuje s úkoly náročnými na výpočetní výkon nebo různými operacemi, které nutně potřebují výpočetní výkon (přehrávání hudby nebo síťové služby), tak je nutné, aby Service vytvořila své vlastní další vlákno, které bude zpracovávat tyto procesy. Tímto se eliminuje zpomalení aplikace a jejích Activit, zvýší se rychlost aplikace a hlavní vlákno aplikace zůstane volné pro interakci uživatele s aplikací.

Třída Service může být ve dvou formách nebo jako jejich kombinace. Verze spouštěná, která může běžet do nekonečna, i když je komponenta, která ji vytvořila, ukončena. Obvykle tento typ Service je jednoduchá jediná operace, která nevrací výsledek své spouštěcí komponentě. Může se jednat tak o typický případ stahování nebo nahrávání souboru přes síťové připojení. Po ukončení této operace by měla byt Service rovněž ukončena buď nějakou komponentou, nebo se ukončí sama.

Druhou verzí je verze vázaná. Tento druh Service nabízí klientské rozhraní, které umožňuje spolupracovat se Service, posílat ji žádosti a data a zpátky získávat data a výsledky. Vázaná forma Service běží jen tak dlouho jak dlouho běží klientská komponenta, která si ji navázala. Více komponent může navázat spojení s touto Service, ale jakmile jsou všechny komponenty ukončeny, dojde k ukončení i dané Service. Pokud chceme vytvořit tento typ Service, je nutné poskytnout rovněž rozhraní, které klient může využít ke komunikaci se Service. Existují tří způsoby, jak definovat toto rozhraní. Prvním je použití třídy Binder. Druhým je použití třídy Messenger a nakonec třetím je využití AIDL. AIDL je využito v aplikaci v této diplomové práci.

Existuje rovněž podtřída Service pod názvem IntentService. Tato třída využívá pracovní vlákno k správě všech požadavků aplikace. Tato správa požadavků zpracovává požadavky postupně, po jednom. Je to tak nejlepší řešení, pokud aplikace nepožaduje zpracovávání mnoha požadavků zároveň.

1.3.4 AIDL

AIDL je jazyk pro definici rozhraní, které klientu a Service umožní komunikovat pomocí IPC. Toto rozhraní pro komunikaci převádí veškeré objekty na primitivní datové typy z důvodu, aby jim rozuměl operační systém. Service tak dokáže obsloužit a zpracovat více požadavků zároveň. Tato funkčnost se hodí pro více vláknovou komunikaci.

Pro využití AIDL v Androidu je nutné vytvořit soubor s koncovkou aidl, kterým definujeme rozhraní. Tohoto rozhraní se pak využívá ke generování abstraktní třídy. Tuto abstraktní třídu pak je nutné použít v třídě Service, kde je nutné implementovat všechny její metody z rozhraní. Pro velkou většinu aplikací není nutné používat AIDL pro vytvoření vázané Service, jelikož to vyžaduje více vláknové schopnosti aplikace a má za následek složitější programování aplikace. Níže uvádím pro ukázku kód ze soubor AIDL z diplomové práce.

```
interface IWifiMessengerCallback {
   void newMessage(String from, String to, String text, String
time);}
```

1.3.5 Application

Application je hlavní třída aplikace. Existuje pouze jediná pro celou aplikaci a sdílí plný životní cyklus aplikace. Spouští se před Activity a Services. Lze k ní přistupovat odkudkoliv v aplikaci. To znamená, že lze přistupovat k jejím proměnným a metodám z jakékoliv Activity nebo kdekoliv, kde máme přístup k objektu Context.

Samotná třída Application běží na pozadí ve vlákně pro uživatelské rozhraní, zatímco třeba třída IntentService běží ve svém odděleném vlastním pracovním vlákně. Určena je pro toho, kdo potřebuje spravovat globální stav aplikace. Lze si vytvořit svou vlastní implementaci pomocí rozšíření této třídy. Pak je nutné přidání android:name s názvem této třídy do Android Manifestu. Instance této třídy je spuštěna, když je spuštěna aplikace. Slouží třeba k uložení globálních proměnných pro celou aplikaci a lze ji využít k průběžné analýze běhu aplikace.

1.3.6 Android Manifest

Každá aplikace musí obsahovat soubor AndroidManifest.xml v hlavní složce programu. Tento soubor obsahuje základní informace o aplikaci pro operační systém, které jsou nutné pro její úspěšné spuštění. Manifest obsahuje unikátní název aplikace, který slouží jako její identifikátor. Dále obsahuje soupis jednotlivých Activit, Services, Broadcast Receiverů a dalších. V Manifestu může být dále uvedeno, které povolení pro práci s chráněnými částmi systému aplikace požaduje. Například povolení pro práci s Bluetooth nebo Wi-Fi.

1.3.7 **Broadcast Receiver**

Broadcast Receiver je třída sloužící k naslouchání akcím a událostem v operačním systému nebo v aplikaci. Každá tato akce nebo událost pošle zprávu aplikaci a ta na ni podle určení reaguje, například výpisem na stavový řádek nebo spuštěním jiné komponenty. Aplikace mohou využívat zprávy systémové nebo vytvářet své vlastní. Tyto zprávy se a události se šíří pomocí třídy Intent. Podobně jako Service ani Broadcast Receiver nemá uživatelské rozhraní. Příklad použití Broadcast Receiveru může být reakce tlačítka na obrazovce na zapnutí Bluetooth. Lze vytvořit Broadcast Receiver, který naslouchá jen konkrétním zprávám. Toho lze dosáhnout pomocí Intent filtru.

1.3.8 **Context**

Context je rozhraní pro globální informace o aplikačním prostředí. Je to abstraktní třída, jejíž realizace je zajištěna Android systémem. Umožňuje přístup ke zdrojům a třídám aplikace, spouštění aktivit, posílání a přijímání Intent zpráv a další. Jak název napovídá, je to kontext současného stavu aplikace nebo objektu. Umožňuje nově vytvořeným objektům pochopit, co se děje.

1.3.9 **Povolení**

Základní aplikace pro Android nemá ve výchozím nastavení žádné oprávnění. To znamená, že nemůže dělat nic, co by mohlo nějakým nepříznivým způsobem ovlivnit běh nebo data zařízení, na kterém aplikace běží. Pokud chceme aplikaci dovolit využívat některé chráněné součásti systému na zařízení, je nutné do Android Manifestu přidat jeden nebo více tagů <uses-permission>, které umožňují aplikaci požádat operační systém o práva při instalaci aplikace.

Aplikaci tak lze buď nainstalovat a spouštět se všemi požádanými právy automaticky, nebo instalaci aplikace zcela odmítnout. V Androidu 6.0 je toto pravidlo ovšem změněno. Změna spočívá v rozšíření možností uživatele Androidu tak, aby aplikace získala po instalaci jen skupinu práv, které nezbytně potřebuje ke svému běhu, tedy které jí uživatel dovolí používat. Aplikace pak může požádat o dodatečná práva při svém běhu, ale uživatel ji může tato práva odepřít.

2 Ad-hoc

Běžná Ad-hoc komunikační síť je typ bezdrátové sítě, která je tvořena dvěma a více rovnocennými prvky a nespoléhá se při komunikaci na přístupové body. Místo toho se každý prvek chová jako samostatný uzel a podílí se na směrování předáváním dat pro jiné uzly, takže určení, které uzly přepošlou data k cíli se provádí dynamicky na základě připojení k síti, její velikosti a délky trasy k tomuto cíli.

Pokud jeden uzel vypadne, nahradí jej jiný, který jinou trasou dokáže předat data cílovému zařízení. Toto směřování se provádí pomocí směrovacích protokolů, kterých je celá řada. Hlavním rozdílem těchto protokolu je způsob, jak mapují síť a jak ukládají cestu k dalším uzlům. Některé ukládají všechny cesty a některé znají jen její část. Existuje jich přes 70 různých druhů. Nejznámější jsou například AODV, DSDV nebo OLSR. Mobilní bezdrátová Ad-hoc síť je pak síť, kde se mobilní zařízení mohou připojovat, odpojovat a přemísťovat bez toho, aby ovlivnili správnou funkčnost sítě.

2.1 Ad-hoc v Androidu

V Androidu je Ad-hoc komunikační síť tvořena jako dočasné mobilní bezdrátové síťové spojení mezi dvěma a více rovnocennými prvky bez přístupového bodu AP. Android umožňuje přímé propojení dvou a více mobilních zařízení mezi sebou pomocí Bluetooth nebo Wi-Fi Direct.

Platforma Android v současné verzi Bluetooth API a Wi-Fi Direct API nedokáže využít plně Ad-hoc způsob komunikace, jelikož neexistuje API, které by se dalo takto využít a vzniká zde rovněž omezení z principu fungovaní Bluetooth a Wi-Fi Direct, kdy tyto technologie umožňují komunikaci pouze ve skupině, kde existuje jeden server, a ostatní zařízení jsou klienti. Celá síť funguje tak, že první spuštěný klient tvoří server, který komunikuje pouze s jednotlivými klienty. Obě technologie neobsahují žádnou implementaci jakéhokoliv směrovacího protokolu pro komunikaci mezi klienty. Veškeré směrování a komunikaci mezi klienty přes server je tedy nutné řešit na úrovni aplikace.

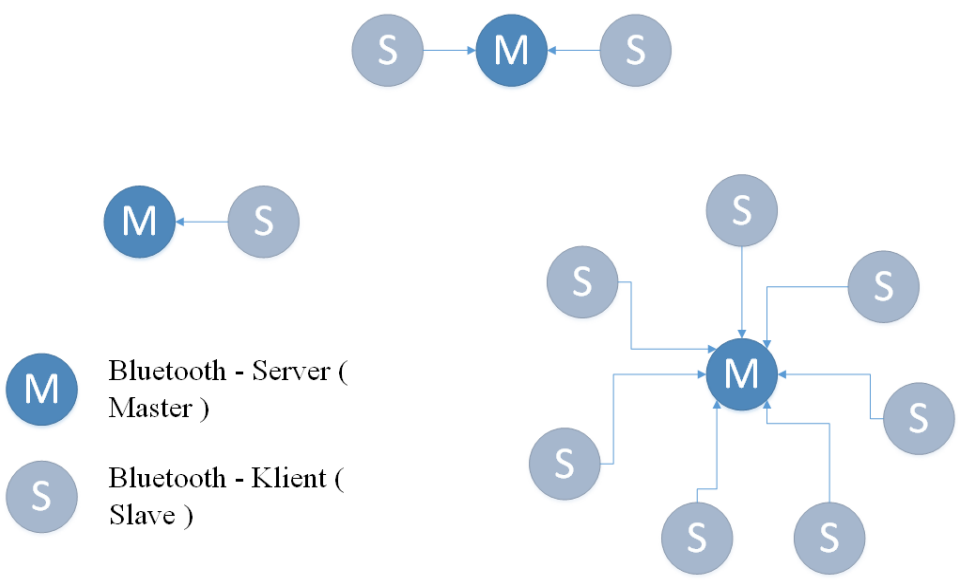
2.1.1 Ad-hoc přes Bluetooth

Propojení zařízení přes Bluetooth je v Androidu realizováno pomocí vyhledání, a v případě zabezpečeného připojení, také spárování zařízení. Jakmile jsou dvě a více zařízení nalezena nebo spárovaná, lze mezi nimi po připojení komunikovat. Při komunikace se vždy jedno ze zařízení chová jako server (Master) a všechny ostatní zařízení se chovají jako klienti (Slaves). Je nutné ovšem upozornit, že ve velké míře záleží na kompatibilitě hardwaru všech zařízení, jestli bude tato komunikace funkční. Rovněž je nutné, aby se předem nastavilo, které zařízení bude serverem. Toto zařízení pak bude naslouchat příchozím žádostem klientů o připojení. Pokud

server vypadne, je jej musí nahradit jiné zařízení a všechna dříve připojená zařízení se musí k tomuto novému serveru znovu připojit.

Hardwarová specifikace Bluetooth a Android API umožňují zařízení, které je serverem, komunikovat až se sedmi klienty zároveň. Klienti nekomunikují mezi sebou, ale komunikace probíhá pouze mezi serverem a jednotlivými klienty. Server však může zajistit přeposlaní komunikace jinému klientovi, avšak toto je nutné řešit v aplikaci, jelikož samotné Bluetooth a Android API tuto funkcionalitu nepodporují. Pokud dojde k zániku serveru, dojde rovněž ke zrušení celé komunikační sítě a je nutné, aby se stalo jiné zařízení serverem. Tento proces však není automatický, je nutné manuální vytvoření nového serveru a opětovné připojení všech klientů.

Důvodem pro omezení na sedm klientů je tříbitová BT_ADDR adresa Bluetooth zařízení. Tříbitová adresa dokáže obsloužit osm zařízení, tedy jeden server a sedm klientů. Až dalších 255 vedlejších zařízení může být neaktivních nebo vyčkávajících, až je server vyzve ke komunikaci. Tento druh sítě se nazývá Piconet a jeho schéma je zobrazeno na obrázku níže.

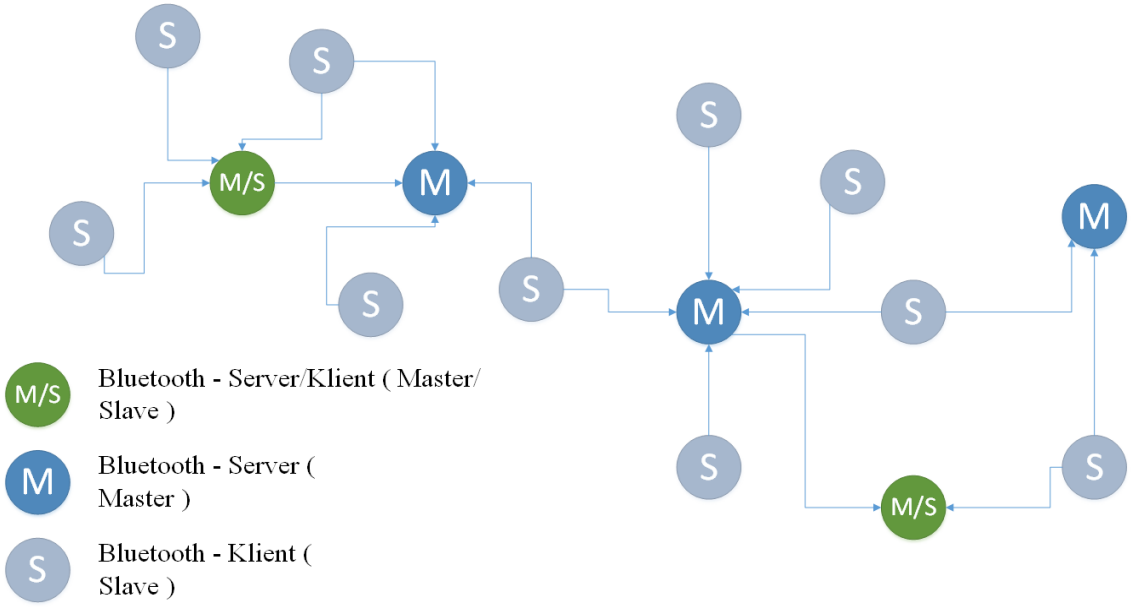


Obrázek 1.2: *Schéma sítě Piconet*

Rozšířenou verzí Piconetu je typ sítě Scatternet. Tato síť se skládá ze dvou nebo více propojených Piconetů. Umožňuje tak komunikaci více než osmi zařízení. Scatternet lze vytvořit tak, že jedno zařízení, které je v Piconetu serverem nebo klientem, bude vykonávat funkci klienta v jiném Piconetu. Toto zařízení, které se pak nachází v obou Piconetech, pak může posílat a přijímat data z obou těchto sítí a umožňuje tím tak jejich propojení.

Funkci a způsob komunikace ve Scatternetu však základní Bluetooth protokol a Android API nepodporují. Je proto nutné veškeré směrování vyřešit opět na straně aplikace v každém zařízení. Tento způsob pak umožňuje spojit několik Piconetů do většího Scatternetu a rozšířit tak

fyzicky velikost sítě za maximální dosah bezdrátové technologie Bluetooth. Na obrázku níže je schéma Scatternetu, které obsahuje několik Piconetů se zařízeními, která fungují jako klient a server ve více Piconetech.



Obrázek 1.3: *Schéma sítě Scatternet*

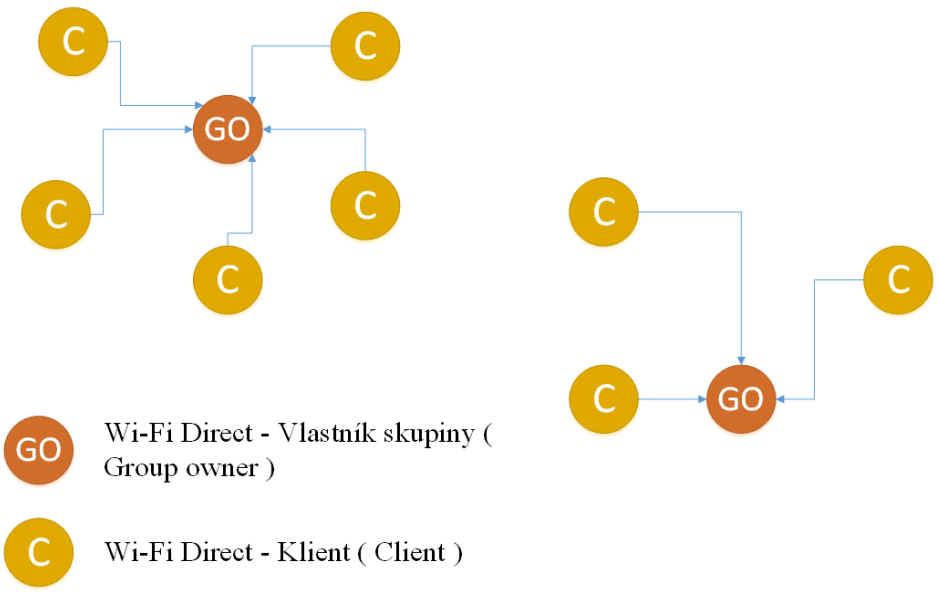
2.1.2 Ad-hoc přes Wi-Fi Direct

Propojit zařízení v Androidu lze kromě Bluetooth i přes Wi-Fi Direct. Podobně jako v Bluetooth, tak rovněž přes Wi-Fi Direct je propojení realizováno pomocí vyhledání a následného automatického spárování zařízení. Po spárování lze zařízení spojit a komunikovat mezi nimi. Komunikace pak probíhá tak, že jedno ze zařízení se stane serverem a vytvoří skupinu. K tomuto vlastníkovi skupiny (Group owner) se pak všechna ostatní zařízení mohou připojit jako klienti (Clients), kteří komunikují pouze s vlastníkem skupiny, nikoliv mezi sebou. Komunikace mezi klienty přes vlastníka skupiny je možná, jen se musí řešit tato komunikace a směrování opět na straně aplikace, jelikož samotné Wi-Fi Direct a jeho API v Androidu tuto funkci nepodporují. Klientů připojených k vlastníkovi skupiny může být v Androidu až 32.

Z obecných specifikací technologie Wi-Fi Direct vyplývá, že lze mít více vlastníků skupiny ve skupině, aby mohli současně komunikovat mezi sebou společně se svými členy. Lze tak tedy udržovat členství ve více skupinách současně. Avšak na Androidu toto neplatí, jelikož implementace Wi-Fi Direct je v Androidu omezená a umožňuje existenci pouze jednoho vlastníka skupiny uvnitř skupiny. Ostatní zařízení jsou pak již jen jeho klienti. Dalším omezením je nemožnost vytvoření propojení mezi jednotlivými skupinami. Pro bližší objasnění zde uvedu příklad.

Máme čtyři zařízení A, B, C, D. Zařízení A a B jsou ve skupině, A je vlastníkem skupiny, B je klient. Zařízení C a D jsou také ve skupině, D je vlastníkem skupiny a C je klient. Představme si, že klient C se chce připojit k vlastníkovi skupiny A a chce se tak stát propojením mezi skupinami. Výsledkem však je, že se toto propojení skupin neuskuteční a nebude fungovat, jelikož když se klient C připojí k vlastníkovi skupiny A, dojde k tomu, že klient C již není nadále ve skupině vlastněné zařízením D, ale patří do skupiny zařízení A. Tedy klient C nemůže nadále komunikovat se zařízením D. Z toho plyne, že pro komunikaci dvou zařízení je nutné, aby patřily do stejné skupiny, jinak komunikace není možná. Tedy zařízení, které je vlastník skupiny, nemůže být klientem v jiné skupině. Zařízení, které je připojené k vlastníkovi skupiny jako klient, nelze připojit do jiné skupiny a fungovat jako jejich propojení.

Pokud zařízení, které je vlastníkem skupiny, zmizí z dosahu klientů nebo dojde k jeho vypnutí, tak skupina automaticky zaniká a je nutné, aby se jiné zařízení stalo vlastníkem skupiny. Tento proces však není automatický, je nutné se opět manuálně z každého zařízení připojit a vytvořit novou skupinu s vlastníkem. Na obrázku níže je jednoduché schéma Wi-Fi Direct sítě. Tato sít obsahuje vždy pouze jednoho vlastníka skupiny.



Obrázek 1.4: *Schéma sítě Wi-Fi Direct*

3 Bluetooth

Obrázek 1.5: *Logo Bluetooth*

Bluetooth je standard pro bezdrátovou komunikaci propojující dvě a více elektronických zařízení. Dokáže propojit mobilní telefon, tablet, osobní počítač, notebook, bezdrátová sluchátka a různé chytré náramky a hodinky. Bluetooth se používá k přenosu informací mezi dvěma nebo více zařízeními, která jsou blízko u sebe.

Technologie Bluetooth pracuje v pásmu 2,4 GHz. Funguje na principu přeskakování několika frekvencí a obsahuje protokoly, které usnadňují rozpoznání a nastavení služeb pro spojení jednotlivých zařízení. Vyskytuje se v několika verzích. Jsou definovány tři výkonové skupiny, s nimiž je umožněna komunikace do vzdálenosti 1–100 metrů. Udávané hodnoty ovšem platí jen ve volném prostoru. Pokud jsou mezi komunikujícími zařízeními překážky, dosah rychle klesá. Přenosová rychlost je různá u každé jednotlivé verze Bluetooth a je možné vytvořit datový spoj symetrický nebo asymetrický, kdy přenosová rychlost při příjmu je vyšší než při odesílání. Jednotlivá zařízení jsou identifikována pomocí své adresy BT_ADDR podobně, jako je MAC adresa u Wi-Fi.

3.1 Verze

Vzhledem k neustálému vývoji nových typů zařízení je nutné Bluetooth udržovat aktuální a přizpůsobit jej současným trendům a požadavkům. Proto jsou jednou za čas představovány nové verze, které se soustřeďují na implementaci nových funkcí a různých vylepšení.

První verzí Bluetooth byla verze 1.0, která byla následována verzí 1.0b. Tyto verze nebyly bez chyb a měly problémy s funkčností. Následující verze 1.1 obsahovala opravy nalezených chyb, byla přidána podpora pro nešifrované kanály a přidán indikátor síly signálu. Rovněž byly stanoveny základní specifikace. Další verze 1.2 přinesla jako hlavní novinku vyšší přenosové rychlosti, rychlejší vyhledávaní a připojování k zařízením a lepší odolnost vůči rušení.

Po verzi 1.2 přišly verze 2.0 a 2.1, které přinesly nové snazší bezpečné párování zařízení a zvýšení rychlosti přenosu. Další verzí se stala verze 3.0, u které došlo opět k zvýšení přenosové rychlosti a k opravě chyb.

Verze 4.0 představila zvýšenou rychlost přenosu a hlavně nízkoenergetickou variantu Bluetooth pro sběr dat od zařízení, která generují málo dat a nepotřebují vysoké datové toky. Hlavní účel této funkce, nazvané Low Energy (LE) je sběr dat od různých senzorů, jako jsou

chytré hodinky, fitness náramky, monitory srdečního tepu a další. Komerčním název této technologie je Bluetooth Smart.

Verze 4.1 je zpětně hardwarově kompatibilní se starší verzi Bluetooth 4.0. Jde pouze o softwarové rozšíření, která vylepšuje podporu LTE a nabízí možnost řetězení zařízení. Poslední verzí je verze 4.2 vydaná v prosinci 2014, která přinesla podporu protokolů pro komunikaci s chytrým příslušenstvím, jako jsou třeba chytré žárovky. Byla také opět zvýšena přenosová rychlost, byla snížena spotřeba energie a byla přidána podpora pro síťovou komunikaci a přístup k internetu prostřednictvím protokolu IPv6 a zlepšeno zabezpečení díky obousměrně šifrovanému přenosu dat. Tyto nové funkce však nejsou podporované starším hardwarem s verzi 4.1 a nejedná se tak pouze o softwarovou aktualizaci.

3.1.1 **Označení verzí**

Bluetooth 4.0 přineslo zejména funkcí Bluetooth Low Energy (LE), díky které mohou některá specifická zařízení pracovat a vysílat přes Bluetooth bez dobíjení několik měsíců a déle. Bluetooth na současných zařízeních, využívajících verzi 4.0 a vyšší, není označeno číslem verze, ale jedním ze tří log. Na zařízeních podporujících Bluetooth lze nalézt loga klasického Bluetooth, Bluetooth Smart Ready a Bluetooth Smart. Tyto loga vysvětlují, co dané zařízení nabízí, a jaké jsou jeho možnosti, vlastnosti a kompatibilita.

Obrázek 1.6: *Loga Bluetooth zařízení*

Kromě základní třídy Bluetooth existuje Bluetooth Smart, do které patří přístroje senzorového typu, jako jsou měřiče srdečního tepu hodinky, různá čidla či krokoměry. Mají slabší baterii a byly vytvořeny jen pro sdílení určitého druhu informace, takže kvůli nižší spotřebě se nepropojí s klasickým Bluetooth, ale dokáží komunikovat pouze s univerzálním novým typem Bluetooth Smart Ready. Zařízení Bluetooth Smart jsou tedy pouze určena k získání konkrétních informací a následně jejich odeslání na produkty Bluetooth Smart Ready, například tablet nebo mobilní telefon.

Bluetooth Smart Ready označuje nejvyšší třídu, která komunikuje se všemi ostatními druhy Bluetooth zařízení. Produkty označené logem Bluetooth Smart Ready lze připojit k milionům zařízení obsahujícím klasické Bluetooth, ale také i k zařízením nesoucí logo Bluetooth Smart. V praxi se jedná o mobilní telefony, tablety, počítače či televize. Takto označené výrobky mohou přijímat data z produktů Bluetooth Smart a následně s nimi pracovat.

3.2 Bluetooth v Androidu

Android zahrnuje podporu Bluetooth stacku a poskytuje tak přístup k funkcionalitě přes své vlastní API postavené nad Bluetooth stackem. Toto API dovoluje aplikacím se připojit bezdrátově k jiným Bluetooth zařízením a umožňuje zařízení komunikovat s jedním nebo více zařízeními. S využitím API může jakákoliv aplikace skenovat zařízení, zjistit spárovaná zařízení, vytvořit spojení přes RFCOMM kanály, připojit se k jiným zařízením přes vyhledání služby, přenést data do a z jiného zařízení a rovněž spravovat více připojení v jednu chvíli. API rovněž dovoluje zařízení s touto podporou, bezdrátově komunikovat a vyměňovat si data s jiným takovým zařízením.

Android podporuje klasické Bluetooth od verze Androidu 2.0. První podporovanou verzi Bluetooth byla verze 2.1. Klasické Bluetooth je určeno pro operace, které jsou energeticky náročné, tedy například operace jako jsou přenášení dat a komunikace mezi zařízeními.

Částečnou podporu nízkoenergetického Bluetooth LE získal Android až s příchodem verze Androidu 4.3. První podporovanou verzí Bluetooth LE byla jeho první verze, která je součástí Bluetooth verze 4.0. Bluetooth LE je oproti klasickému Bluetooth výhodné u zařízení, která nepřenáší data často, přenáší jen malé objemy dat a nemají vysoké energetické požadavky. Typickými zařízeními jsou různé fitness náramky nebo chytré hodinky propojené s mobilním telefonem.

S postupným vývojem Androidu docházelo k přidávání podpory novějších verzí Bluetooth. Podporu Bluetooth 3.0 přinesl Android 4.0. Android 4.2 přinesl podporu verze Bluetooth 4.0 a Android verze 5.0 již podporoval Bluetooth 4.1 včetně dokončení plné podpory nízkoenergetické varianty LE. Aktuální Android 6.0 podporuje zatím poslední a nejaktuálnější Bluetooth 4.2. Příkladem mobilních telefonů, které obsahují Android 6.0 a softwarovou a hardwarovou podporu Bluetooth 4.2, jsou telefony Huawei Nexus 6P a LG Nexus 5X. Nutno ovšem podotknout, že nové funkce Bluetooth 4.2 nejsou podporované staršími čipy v telefonech, čili je budou moct využívat hlavně nová zařízení. Neopakuje se tak situace, která nastala po představení specifikace 4.1, jež byla aktualizována softwarově a tudíž zpětně kompatibilní s velkou částí přístrojů.

V následujících podkapitolách budu popisovat práci s klasickým Bluetooth, které jsem využil v této diplomové práci. Všechny kódy a popisovaná funkčnost je ukázkovým obsahem a je vyjmuta z aplikace diplomové práce.

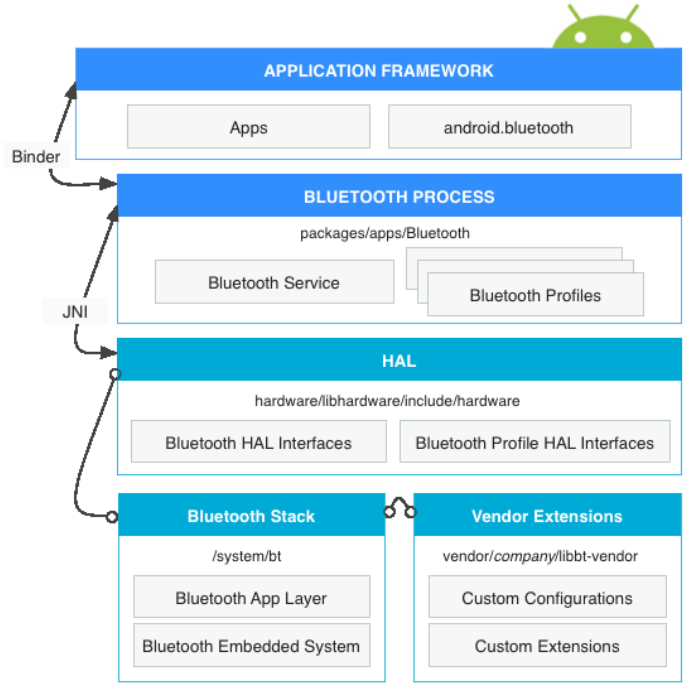
3.2.1 Stack

Bluetooth stack je sada ovladačů a rozhraní, které poskytují počítačovým a mobilním aplikacím přístup k Bluetooth protokolům bezdrátové technologie Bluetooth. Tato mezivrstva je tedy jakýsi řídící software, který komunikuje s hardwarem Bluetooth a na kterým je postaveno Bluetooth API Androidu.

Android poskytuje svůj vlastní Bluetooth stack, který je rozdělen do dvou vrstev. První základní vrstva zahrnuje základní funkcionalitu Bluetooth. Druhou vrstvou je Bluetooth aplikační vrstva. Obrázek níže ukazuje strukturu Bluetooth stacku. Aplikační framework obsahuje Bluetooth API, které spolupracuje s hardwarem pomocí stacku. Pro operační systém Linux je od verze jádra 2.4.6 k dispozici oficiální Bluetooth stack BlueZ, původně vyvíjený firmou Qualcomm. Vzhledem k tomu, že samotný Android je postavený na Linuxu, tak BlueZ využíval Google v Androidu po řadu verzí od první verze 1.0 až po verzi 4.1.

Z řady blíže nespecifikovaných důvodů Google ke konci roku 2012 vyměnil Bluetooth stack v Androidu. Z původní verze BluZ od společnosti Qualcomm, přešel na open source BlueDroid od společnosti BroadCom. Tato výměna proběhla při vydání Androidu 4.2. Z tohoto důvodu dochází občas k různým případům nekompatibility a nemožnosti komunikace mezi aplikacemi nainstalovaných mezi zařízeními, které obsahují různé verze stacku.

Platforma Android zahrnuje podporu pro Bluetooth stack, který umožňuje zařízení bezdrátově komunikovat a posílat data ostatním zařízením. Aplikační framework Androidu poskytuje přístup k Bluetooth funkcím skrze Android Bluetooth API. Tyto API umožnují aplikacím se bezdrátově připojit k jiným Bluetooth zařízením a poskytuji tak možnost komunikace mezi jedním a jedním nebo více zařízeními.



Obrázek 1.7: *Architektura Bluetooth v Androidu*

3.2.2 Android API v aplikaci

V diplomové práci jsem využil klasické Bluetooth. V následujícím textu tedy budu popisovat pouze klasické Bluetooth, nikoliv Bluetooth LE z balíčku android.bluetooth.le [17].

Všechny hlavní třídy a rozhraní pro práci s klasickým Bluetooth lze nalézt v balíčku tříd android.bluetooth [13]. Tyto třídy a rozhraní umožňují například správu nastavení Bluetooth modulu, hledání viditelných nebo spárovaných zařízení nebo připojení k zařízení a přenos dat mezi nimi. V následujících řádcích popíšu nejdůležitější třídy a rozhraní z tohoto balíčku.

Hlavní třídou práci s Bluetooth je třída BluetoothAdapter. Tato třída reprezentuje Bluetooth na zařízení a zahrnuje funkce a metody pro práci s ním. Třída umožňuje vyhledání jiných Bluetooth zařízení. Rovněž lze zjistit seznam spárovaných zařízení, najít zařízení BluetoothDevice pomocí BT_ADDR adresy a vytvořit BluetoothServerSocket pro komunikaci s jinými zařízeními.

Třída BluetoothDevice představuje vzdálené Bluetooth zařízení. Lze se pomocí této třídy požádat o spojení s jiným zařízením pomocí BluetoothSocketu a nebo lze zjistit informace o vzdáleném zařízení jako je jeho jméno, párování, třída atd.

Třída BluetoothSocket představuje rozhraní pro Bluetooth socket, který je velmi podobný jako TCP klientský socket. Slouží jako připojovací bod, který umožňuje aplikacím výměnu dat s jiným Bluetooth zařízením pomocí InputStreamu a OutputStreamu.

Třída BluetoothServerSocket je velmi podobná TCP serverovému socketu. Reprezentuje otevřený serverový socket, který čeká na příchozí požadavky. Pro spojení dvou Android zařízení, jedno zařízení musí otevřít tento socket pomocí této třídy. Když Bluetooth zařízení vytvoří požadavek na připojení k serveru, tak instance třídy BluetoothServerSocket vrátí BluetoothSocket zařízení, pokud je připojení k zařízení povoleno.

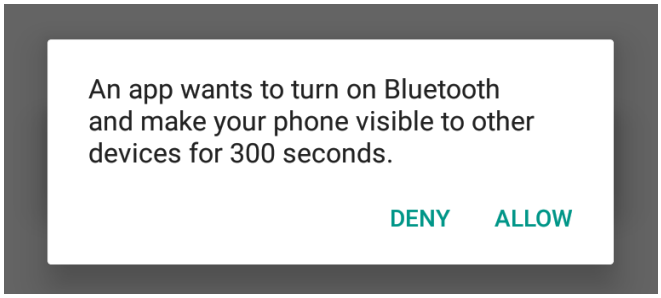
3.2.3 Nalezení a párování zařízení

Pro Bluetooth v Androidu obecně platí, že nelze vyhledat zařízení, které obsahuje námi hledanou aplikaci se službou. Vyhledávání vždy nalezne všechna zařízení, která obsahuji podporu Bluetooth komunikace a která jsou viditelná nebo spárovaná a dosažitelná.

Pro získání přístupu k Bluetooth na zařízení je nutné jako první získat instanci třídy BluetoothAdapter. Pokud BluetoothAdapter nevrátí null, zařízení podporuje Bluetooth a lze s ním dále pracovat. Získat BluetoothAdapter lze dvěma způsoby, jejichž použití se odvíjí od verze API, kterou používáme v aplikaci. Na níže uvedeném kódu lze vidět rozdíl v metodách pro Android 4.3 JELLY BEAN MR2 a vyšší a Android 4.2 JELLY BEAN MR1 a nižší.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR2)
{
BluetoothManager mBluetoothManager = (BluetoothManager)
getSystemService(Context.BLUETOOTH_SERVICE);
            mBluetoothAdapter = mBluetoothManager.getAdapter();}
else { mBluetoothAdapter =
BluetoothAdapter.getDefaultAdapter();}
```

Bluetooth zařízení s operačním systémem Android je v základním nastavení neviditelné pro všechna ostatní nespárovaná zařízení. Zviditelnit lze zařízení pomocí níže uvedeného kódu, který vytvoří dialogové okno, ve kterém uživatel může, nebo nemusí povolit zviditelnění zařízení. Pokud povolí zviditelnění zařízení, dojde rovněž k zapnutí Bluetooth. Hodnota proměnné duration znamená, na jak dlouho bude zařízení vždy viditelné pro všechna ostatní nespárovaná zařízení. Pokud nastavíme 0, zařízení bude vždy viditelné. Jakákoliv jiná nezáporná hodnota znamená dobu, po kterou bude zařízení viditelné. Uvedena hodnota je v sekundách.



Obrázek 1.8: *Povolení zviditelnění Bluetooth zařízení*

```
if (mBluetoothAdapter.getScanMode() !=
BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
        Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);

discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_
DURATION, duration);

discoverableIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        mContext.startActivity(discoverableIntent);
    } else { getMessage(mContext,
mContext.getResources().getString(R.string.discoverable_already)
);}
```

Samotné vyhledávání všech viditelných zařízení lze spustit pomocí metody BluetoothAdapter.startDiscovery(). Tato metoda využívá registrovaný Broadcast Receiver s Intent filtrem uvedeným na kódu níže.

```
IntentFilter filter = new IntentFilter();
filter.addAction(BluetoothDevice.ACTION_FOUND);
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
```

V Broadcast Receiveru lze nalézt tyto viditelná zařízení pak pomocí tohoto kódu, který ukládá zařízení do seznamu na obrazovce.

```
if (BluetoothDevice.ACTION_FOUND.equals(action)) {
    BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
    if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
        mArrayList.add(new
ChatList(ContextCompat.getColor(getActivity(),
R.color.md_orange_500), device.getName(), device.getAddress(),
device.getBluetoothClass().toString(),
Methods.getDeviceStatus(getActivity(), device.getBondState())));
        mAdapter.notifyItemInserted(mArrayList.size() - 1);
    }
```

Již spárovaná a uložená zařízení lze nalézt a přidat do seznamu na obrazovce pomocí níže uvedeného kódu. Tento kód nalezne všechna spárovaná zařízení, jejichž záznam je uložen v operačním systému. Neplatí však, že se k nim lze připojit, jelikož daná zařízení nemusí být v dosahu zařízení.

```
Set<BluetoothDevice> pairedDevices =
mBtAdapter.getBondedDevices();
if (pairedDevices.size() > 0) {for (BluetoothDevice device :
pairedDevices) { mArrayList.add(new
ChatList(ContextCompat.getColor(getActivity(),
R.color.md_orange_500), device.getName(), device.getAddress(),
device.getBluetoothClass().toString(),
Methods.getDeviceStatus(getActivity(), device.getBondState())));
mAdapter.notifyItemInserted(mArrayList.size() - 1);}}
```

3.2.4 Připojení jako klient a zabezpečení

Jak víme z předchozí kapitoly, pro Bluetooth v Androidu platí, že nelze vyhledat zařízení s konkrétní aplikaci, ale jen všechna s podporou Bluetooth komunikace, která jsou viditelná nebo spárovaná. Pro odlišení jednotlivých aplikací však Bluetooth na Androidu používá UUID, které musí být jedinečné pro zabezpečenou a nezabezpečenou komunikaci aplikace. Dvě stejné aplikace musí obsahovat stejnou verzi UUID, aby mohla navzájem komunikovat. Na internetu lze nalézt generátory UUID [7], které vygenerují tento náhodný kód pro aplikaci.

UUID je Universally Unique Identifier (UUID). Je to standardizovaný 128 bitový řetězec pro unikátní identifikaci Bluetooth aplikace. Příklady UUID kódů jsou uvedeny níže.

```
public static final UUID BT_UUID_SECURE =
UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");

public static final UUID BT_UUID_INSECURE =
UUID.fromString("8ce255c0-200a-11e0-ac64-0800200c9a66");
```

Pokud je Bluetooth zařízení viditelné nebo spárované a dosažitelné, lze jej nalézt a lze se k němu připojit. Android umožňuje dvě možnosti připojení k jiným zařízením. První možností je vytvoření nezabezpečeného připojení k zařízení. Pro toto připojení stačí, pokud známe BT_ADDR adresu zařízení. Zařízení se spojí a jejich komunikace bude funkční, aniž by bylo nutné je jakkoliv párovat.

Druhou možností je vytvoření zabezpečeného připojení, kdy je nutné opět znát BT_ADDR adresu zařízení. Při připojování k zařízení dojde k jejich párování a zobrazení náhodně vygenerovaného PIN kódu na obou zařízeních. Tento PIN kód je generován pomocí algoritmu E22. Využití generování je však upraveno, pokud jedno zařízení obsahuje pevný PIN kód. V tomto případě se využije tento pevný PIN a nikoliv náhodně generovaný. Uživatelé mají díky použití PIN kódu možnost párování a připojovaní k zařízení odmítnout.

Na kódu níže lze vidět obě metody z třídy BluetoothSocket pro zabezpečené a nezabezpečené spojení a získání BluetoothDevice z BT_ADDR adresy.

```
BluetoothDevice device = adapter.getRemoteDevice(mac);

BluetoothSocket socket;

if (secure == 0) { socket =
device.createInsecureRfcommSocketToServiceRecord(Codes.BT_UUID);

            } else { socket =
device.createRfcommSocketToServiceRecord(Codes.BT_UUID);}

            socket.connect();
```

Pro spárovaná zařízení pomocí zabezpečeného připojení platí, že po spárování si obě ukládají svou BT_ADDR adresu do operačního systému. Není pak nutné zařízení znovu vyhledávat a znovu se párovat, ale lze se navzájem připojovat a odpojovat kdykoliv.

Jakmile jsou dvě Android zařízení s podporou Bluetooth spárovaná, lze tyto zařízení kdykoliv k sobě připojovat a odpojovat. To platí až do té doby, dokud uživatel Android zařízení se nerozhodne vymazat uloženou BT_ADDR adresu z operačního systému na jednom zařízení. Pak je nutné provést znovu proces párování na tomto zařízení.

Pro zařízení, která se připojují k sobě nezabezpečeně, naopak nedochází k ukládání BT_ADDR adresy a je nutné je vždy znovu vyhledat a připojit se k nim.

3.2.5 Tvorba serveru a komunikace

Pro vyhledání aplikací, které jsou identifikovány pomocí UUID se používá protokol SDP. Operační systém Android pak, pro následnou komunikaci přes klasické Bluetooth, používá protokol RFCOMM v jeho zabezpečené a nezabezpečené verzi. Používá se pro připojení, komunikaci a přenosu dat s jiným zařízením.

Klienti v Androidu komunikují po připojení pouze se serverem. Pokud chtějí komunikovat s dalšími klienty, je nutné zajistit tuto komunikaci přes server, který ji přepošle správnému klientovi. Řešení je tedy možné jen na úrovni aplikace, protože Bluetooth neobsahuje tuto funkcionalitu. Komunikace mezi klientem a serverem je v Bluetooth šifrována pomocí šifry E0 a od Bluetooth 4.0 je šifrovaná pomocí metody AES se 128 bitovým šifrováním.

Jak z předchozích kapitol víme, klient se může k serveru připojit pomocí zabezpečeného a nezabezpečeného připojení. To ovšem znamená, že na straně serveru musí existovat metoda, která dokáže naslouchat příchozí žádosti o připojení. A musí existovat jak v zabezpečené, tak v nezabezpečené verzi.

Ve třídě BluetoothServerSocket lze nalézt metody, které používají RFCOMM protokol, pro zabezpečené a nezabezpečené připojení klientů k serveru. Tyto metody jsou uvedeny na kódu níže.

```
BluetoothServerSocket serverSocket;
if (secure == 0) {
serverSocket =
mBluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord
("server", Codes.BT_UUID);
            } else { serverSocket =
mBluetoothAdapter.listenUsingRfcommWithServiceRecord("server",
Codes.BT_UUID);}
```

3.2.6 **Povolení**

Každá aplikace musí pro práci s klasickým Bluetooth obsahovat v Android Manifestu tyto povolení. Bez těchto povolení tedy aplikace nebude fungovat.

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission
android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Od verze Androidu 6.0 s API 23 došlo k odstranění metod pro přístup k BT_ADDR adrese zařízení. Metoda BluetoothAdapter.getAddress() nyní vrací konstantní hodnotu 02:00:00:00:00:00.

Pokud zařízení s Androidem 6.0 se pokusí vyhledat zařízení v okolí, tak je tato operace viditelná pro okolní zařízení jako vyhledávaní z náhodné BT_ADDR adresy.

Pro přistup k BT_ADDR adresám ostatních zařízení v okolí, je nutné, aby aplikace obsahovala dále tyto povolení. Je nutné o tyto povolení požádat za běhu aplikace. Bez potvrzení povolení uživatelem nelze pak vyhledat okolní zařízení a pracovat s Bluetooth. Na zařízení s Androidem 6.0.1 však aplikace nefunguje ani přes použití těchto povolení.

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

4 Wi-Fi Direct

Obrázek 1.9: *Logo Wi-Fi Direct*

Wi-Fi Direct je bezdrátový Wi-Fi standard, který umožňuje přímé propojení dvou nebo více zařízení bez nutnosti bezdrátového přístupového bodu AP jako v běžných sítích. Je tedy jakýmsi ekvivalentem k technologii Bluetooth a Bluetooth LE, oproti které však nabízí vyšší rychlost přenosu, větší dosah signálu a hlavně lepší zabezpečení a šifrování přenosu pomocí WPA2. Využívá klasickou bezdrátovou komunikační technologii Wi-Fi, která k přenosu dat vzduchem využívá frekvenční pásmo od 2,4 GHz do 5 GHz. Zařízení podporující Wi-Fi Direct mezi sebou komunikují v režimu Ad-hoc. Při prvním připojení Wi-Fi Direct zařízení dojde k určení, které zařízení bude sloužit jako přístupový bod neboli vlastník skupiny. Všechna ostatní zařízení se pak po připojení automaticky stávají klienty vlastníka skupiny. Wi-Fi Direct zařízení lze použít pro přenos dat a komunikaci mezi zařízeními nebo pro připojení k internetu. Výhodou Wi-Fi Direct je možnost propojit zařízení od různých výrobců. Wi-Fi Direct zařízení lze propojit buď dvě, nebo lze připojit k jednomu zařízení více dalších a ne všechny tyto zařízení musí mít podporu Wi-Fi Direct. K jednomu zařízení s funkcí Wi-Fi Direct se tedy může připojit zařízení s klasickým Wi-Fi bez funkce Direct. Nevýhodou je, při použití v mobilních zařízeních, vyšší spotřeba energie na rozdíl od Bluetooth nebo Bluetooth LE. Wi-Fi Direct najdeme v mobilních telefonech, tabletech, noteboocích, ale také i v herních konzolích a v mnoha dalších zařízeních. Všude tam umožňuje jednoduché a rychlé přenosy a komunikaci mezi kompatibilními zařízeními.

4.1 Wi-Fi Direct v Androidu

Wi-Fi Direct, jinak také Wi-Fi peer to peer (P2P), umožňuje zařízením s potřebným hardwarem, se připojit přímo k sobě přes Wi-Fi bez přístupového bodu. Wi-Fi Direct je plně podporován v Androidu od verze 4.0. Tato verze operačního systému Android umožňuje vyhledání Wi-Fi Direct zařízení přímo, tedy nelze ověřit, zda obsahují námi hledanou aplikaci. K vylepšení a přidání dalších funkcí došlo v Androidu 4.1, který přidal možnost vyhledání konkrétní aplikace se službou běžící nad Wi-Fi Direct.

Uživatelům bez softwarové podpory Wi-Fi Direct stačí k přímému přenosu dat mezi několika zařízeními, když tuto certifikaci bude splňovat pouze jeden z nich, který bude plnit

funkci vlastníka skupiny, na který se budou následně ostatní klienti připojovat. Pro připojení je však nutné znát jméno skupiny a heslo. Toto ovšem neplatí, pokud zařízení nabízí nějakou službu. V tomto případě je nutné, aby každé zařízení obsahovalo podporu Wi-Fi Direct služeb.

S použitím Wi-Fi Direct Android API lze vyhledat a připojit se k jinému zařízení. Pak lze komunikovat přes rychlejší rozhraní a s vyšším dosahem než přes Bluetooth. Využití lze najít pro aplikace, které sdílejí mnoho dat mezi uživateli. Například různé hry nebo sdílení dokumentů a fotografií. Je-li připojení P2P vytvořeno přes Wi-Fi Direct, zařízení nadále udržuje přes mobilní nebo jakékoliv jiné dostupné sítě své připojení k internetu.

Wi-Fi Direct Android API trpí velkým množstvím chyb a nestabilitou. To je taky důvod, proč v obchodě Google Play existuje jen minimum aplikací, které jej využívají.

V následujících podkapitolách budu popisovat práci s Wi-Fi Direct, které jsem využil v této diplomové práci. Všechny kódy a popisovaná funkčnost je ukázkovým obsahem a je vyjmuta z aplikace diplomové práce.

4.1.1 **Android API v aplikaci**

Pro programování aplikace, která využívá možností Wi-Fi a Wi-Fi Direct lze využít několik API z Android SDK. Tyto API v následujících řádcích popíšu a uvedu jejich účel a možnosti. Také uvedu, které API jsem využil v diplomové práci.

Pro práci s Wi-Fi na mobilním zařízení jsem využil API z balíčku android.net.wifi [14]. Obsahuje třídy a rozhraní, které umožnují získávat informace o stavu Wi-Fi modulu v zařízení a také informace o připojení k síti a internetu. Hlavní třídou je WifiManager, která umožňuje kompletní správu a práci s modulem Wi-Fi v zařízení. Dále obsahuje třídu WpsInfo, která umožňuje Wi-Fi zabezpečené spojení a nastavení WPS.

Pro vytváření a hledání Wi-Fi služby lze využít balíček android.net.nsd [18], který obsahuje API, které dokáže naleznout Wi-Fi službu na lokální síti. Bohužel je toto API omezeno na vytvoření a komunikaci se službou NSD pouze s aplikacemi na zařízeních, která jsou připojena ke stejné lokální Wi-Fi síti. Z toho důvodu jsem toto API v diplomové práci nevyužil.

Další využité API je z balíčku android.net.wifi.p2p [15]. Toto API obsahuje třídy a rozhraní pro vytváření peer-to-peer (P2P) spojení přes Wi-Fi Direct. Hlavní třídou z tohoto balíčku je třída WifiP2pManager, která slouží pro správu Wi-Fi P2P připojení. Obsahuje například metody pro nalezení dostupných zařízení nebo pro nastavení připojení k těmto zařízením. Dále tato třída obsahuje několik rozhraní. Například rozhraní WifiP2pManager.ActionListener() slouží k potvrzení úspěchu operace. Nebo rozhraní WifiP2pManager.GroupInfoListener(), díky kterému lze získat informace o vytvořené skupině. Dále jsem využil rozhraní WifiP2pManager.ConnectionInfoListener(), přes které lze získat informace o spojení.

Třída WifiP2pConfig obsahuje konfiguraci nastavení spojení. Třída WifiP2pDevice obsahuje informace o zařízení. Lze zjistit, jestli zařízení nabízí určitou formu WPS zabezpečení nebo zda je vlastníkem skupiny. Třída WifiP2pDeviceList reprezentuje seznam připojených zařízení. V této třídě jsou metody pro získání seznamu připojených zařízení nebo informace o konkrétním zařízení. Třída WifiP2pGroup obsahuje údaje o skupině. Skupina se skládá z jednoho vlastníka skupiny a z jednoho nebo více klientů. Obsahuje metody pro získání seznamu klientů, získání jména a hesla skupiny. Lze rovněž zjistit, kdo je vlastníkem skupiny a ověřit tuto informaci. Třída WifiP2pInfo obsahuje informace o Wi-Fi P2P skupině. Lze zjistit, kdo je vlastníkem skupiny, jeho MAC adresu a jestli došlo k vytvoření skupiny. Třída WifiP2pManager.Channel připojuje aplikaci k API a většina metod používá instanci této třídy jako parametr.

Poslední použité API lze nalézt v balíčku android.net.wifi.p2p.nsd [16]. Toto API obsahuje třídy, které umožňují vytvoření P2P služby, která poběží nad Wi-Fi Direct. Pouze zařízení, které obsahují aplikaci s touto službou, pak jsou schopná se navzájem nalézt a připojit se k sobě. Třídy z API s názvy WifiP2pDnsSdServiceInfo a WifiP2pServiceInfo slouží k vytvoření služby, která běží nad Wi-Fi Direct. Třídy WifiP2pDnsSdServiceRequest a WifiP2pServiceRequest slouží pak k vyhledávání této vytvořené služby. A nakonec třída WifiP2pUpnpServiceInfo slouží k uložení informací o službě UPnP a třída WifiP2pUpnpServiceRequest slouží k jejímu nalezení.

4.1.2 Nalezení zařízení nebo služby

Pro Wi-Fi Direct v Androidu obecně platí, že lze vyhledat všechna zařízení, které podporují Wi-Fi Direct nebo obsahují námi hledanou aplikaci se službou nad Wi-Fi Direct. Vyhledávání tak vždy nalezne všechna spárovaná nebo nespárovaná zařízení se službou nebo bez služby.

Přístup k Wi-Fi v zařízení lze v Androidu získat pomocí třídy WifiManager a její metody z Contextu s názvem getSystemService(). Na kódu níže lze vidět, jak lze získat tento přistup. Prefix Context před metodou a parametrem není nutné psát, pokud pracujeme přímo v Activity. Tento přistup nám pak umožnuje zapnout a pracovat s Wi-Fi a následně i s P2P API.

```
WifiManager mWifiManager = (WifiManager) Context.
getSystemService(Context.WIFI_SERVICE);
```

Aby zařízení, které obsahující stejnou aplikaci, mohly pomocí této aplikace spolu komunikovat pomocí Wi-Fi Direct, je nutné, aby navzájem byly schopné se detekovat a nalézt případně své aplikací vytvořené služby. Tato detekce a tvorba služby je možná třemi způsoby.

Prvním způsobem je použití API NSD, neboli Network Service Discovery. Toto API z balíčku android.net.nsd umožňuje aplikaci vytvoření služby a nalezení Wi-Fi Direct zařízení,

které obsahuje aplikaci s touto službou. Omezením API je však možnost nalezení pouze takových zařízení, která jsou připojená na stejné lokální Wi-Fi síti. To značně omezuje možnosti aplikace stavěné nad tímto API, jelikož je nutné mít vždy zařízení připojená k Wi-Fi síti.

Druhý způsob je využití P2P API, které se nachází v balíčku android.net.wifi.p2p. Toto slouží k přímému hledání pouze samotných zařízení, které podporují Wi-Fi Direct. Tyto zařízení nemusí být připojené k žádné lokální Wi-Fi síti, neposkytují žádnou službu a nelze tedy rozeznat, které z těchto zařízení, má námi hledanou, spuštěnou aplikaci. Lze tedy nalézt jen všechny zařízení, která podporují Wi-Fi Direct, nikoliv jen ta, která obsahují aplikaci se službou.

Třetím a posledním způsobem je využití balíčku android.net.wifi.p2p a zároveň android.net.wifi.p2p.nsd. Toto P2P API poskytuje, stejně jako druhý způsob, hledaní Wi-Fi Direct zařízení mimo stejnou lokální síť. Ale s tím rozdílem, že nalezne jen ty zařízení, na kterých běží vytvořená P2P služba nějaké konkrétní aplikace vytvořená pomocí P2P API. API rovněž nabízí možnost tvorby této detekovatelné služby a umožnuje práci s touto službou.

V této diplomové práci jsem využil třetí možnost, která je dostupná od Androidu 4.1. V další kapitole budu popisovat pouze tento způsob detekce a vytvoření služby poskytované aplikací na Wi-Fi Direct zařízení.

4.1.3 Vytvoření a detekce služby

V této kapitole popíšu, jak lze v aplikaci detekovat a vytvořit poskytovanou službu, která běží nad Wi-Fi Direct. Pro aplikaci, která takto běží na zařízení, platí, že je detekovatelná pouze stejnou aplikací se stejnou službou na jiném zařízení.

Parametry služby se nastavují pomocí metody newInstance() ze třídy WifiP2pDnsSdServiceInfo. Tato metoda obsahuje parametry instanceName, serviceType a Map<String, String>. Do prvního parametru patří název služby, který musí obsahovat prefix _ před názvem služby. Druhý parametr musí obsahovat _presence._tcp nebo _presence._ipp. Tento parametr znamená typ služby. Obsahem třetího parametru je objekt Map s potvrzovacím stavem služby. Níže uvádím jako příklad část kódu z třídy WifiActivity v diplomové práci.

```
Map<String, String> record = new HashMap<>();
record.put(TXT_RECORD_PROP_AVAILABLE, "visible");
mServiceInfo =
WifiP2pDnsSdServiceInfo.newInstance(SERVICE_INSTANCE,
SERVICE_REG_TYPE, record);
        mManager.addLocalService(mChannel, mServiceInfo, new
WifiP2pManager.ActionListener() {
            @Override
```

```
            public void onSuccess() {
mServiceBroadcastingHandler.postDelayed(mServiceBroadcastingRunn
able,SERVICE_BROADCASTING_INTERVAL);}
            @Override
            public void onFailure(int error) {}
        });
```

Objekt mServiceInfo z ukázkového kódu je pak parametrem metody WifiP2pManager.addLocalService(). Tato metoda slouží k vytvoření detekovatelné služby.

Detekce a nalezení služby naopak probíhá v metodě setDnsSdResponseListeners() ze třídy WifiP2pManager. Prvním parametrem této metody je objekt WifiP2pChannel. Druhým parametrem může být rozhraní WifiP2pManager.DnsSdServiceResponseListener(), jenž umožňuje v metodě onDnsSdServiceAvailable() nalézt všechny klienty, kteří pracují s námi hledanou službou. Níže uvádím opět ukázkový kód z třídy WifiActivity z diplomové práce.

```
mManager.setDnsSdResponseListeners(mChannel,
        new WifiP2pManager.DnsSdServiceResponseListener() {
            @Override
            public void onDnsSdServiceAvailable(String
instanceName, String registrationType, WifiP2pDevice srcDevice)
{
                if
(instanceName.equalsIgnoreCase(SERVICE_INSTANCE)) {
                    WifiAvailableFragment fragment =
(WifiAvailableFragment) adapter.getItem(0);
                    fragment.add(instanceName, registrationType,
srcDevice);
                }
            }
        }, new WifiP2pManager.DnsSdTxtRecordListener() {
            @Override
            public void onDnsSdTxtRecordAvailable(String
fullDomainName, Map<String, String> record, WifiP2pDevice
device) {
            }
        });
```

4.1.4 Párování, připojení k zařízení a zabezpečení

Pokud zařízení nalezne jiné P2P zařízení nebo detekuje P2P službu poskytovanou aplikací na jiném zařízení, může se pokusit k tomuto zařízení připojit.

Při každém prvním a úspěšném připojení k novému zařízení, dojde k automatickému párování zařízení. Toto párování se provádí pouze jednou a dojde k němu, jen pokud se zařízení úspěšně poprvé propojí s druhým zařízením. Pokud jsou zařízení spárovaná, mohou mezi sebou komunikovat a kdykoliv se k sobě libovolně připojovat a odpojovat již bez ověřování zabezpečení WPS.

Toto párování funguje na podobném principu jako párování dvou Bluetooth zařízení. A stejně jako v Bluetooth, ani dvě Wi-Fi Direct Android zařízení nemohou spolu komunikovat, pokud nejsou spárované. Operační systém na obou párovaných zařízeních si pak uloží MAC adresu druhého zařízení do paměti a tam zůstane, dokud ji uživatel nevymaže.

Při prvním párování zařízení se v průběhu připojování ověřuje zabezpečení pomocí WPS. Připojit se lze k jinému zařízení pomocí metody WifiP2pManager.connect(). Jedním z parametrů této metody je instance třídy WifiP2pConfig, jenž obsahuje proměnnou wps.setup. Tato proměnná udává typ WPS zabezpečovací metody. Lze zvolit statické proměnné z třídy WpsInfo a to buď PBC, DISPLAY, KEYPAD, LABEL nebo poslední možnost, kterou je INVALID. Hodnota PBC se rovná hodnotě 0 a znamená automatické párování a připojení zařízení bez autentizace. Tedy jakmile je zařízení nalezeno, lze se k němu připojit. Hodnota DISPLAY se rovná hodnotě 1 a znamená, že hodnota wps.pin je zobrazena na zařízení, ke kterému se připojujeme. Hodnota KEYPAD se rovná hodnotě 2 a znamená, že je nutné zadat stejný pin na obou propojovaných zařízeních. Hodnota LABEL znamená, že pin kód je zadán na zařízení a na zařízení, ke kterému se připojujeme je pouze zobrazen. Hodnota INVALID se rovná hodnotě 5 a nastaví se, pokud nastavíme hodnotu, která je z jiného rozsahu než 0 až 4.

Po úspěšném ověření zabezpečení dojde k spárování mobilních telefonů, uložení MAC adres zařízení do paměti telefonu a k vzájemnému připojení zařízení.

4.1.5 Vytvoření skupiny a komunikace

V průběhu připojování dochází k určení, které zařízení se stane vlastníkem skupiny. Pokud se zařízení připojuje k vlastníkovi skupiny, stane se automaticky jeho klientem. Pokud ovšem není určeno, které zařízení je vlastník skupiny, dojde k jeho určení pomocí určení hodnoty proměnné groupOwnerIntent, kterou obsahuje třída WifiP2pConfig. To znamená, že v Androidu lze, při připojování zařízení k sobě, vynutit, kdo se stane vlastníkem skupiny a vytvoří skupinu. Proměnná groupOwnerIntent má číselný rozsah od 0 do 15, přičemž 0 znamená, že zařízení se stane klientem a 15 vlastníkem skupiny. Ostatní čísla od 1 do 14 jen zvyšují nebo snižují pravděpodobnost.

Pokud vlastník skupiny má nastavenou hodnotu 15, tak všechny ostatní zařízení pak musí obsahovat hodnotu nižší než je 15, aby se staly klienty vlastníka skupiny, jinak se k němu

nepřipojí. Pokud obě zařízení obsahují hodnotu 0, tak se jako vlastník skupiny náhodně určí jedno ze zařízení.

Pokud máme aplikaci, která neposkytuje službu, ale jen běžný Wi-Fi Direct přistup na zařízení, lze vytvořit skupinu pro zařízení, jenž Wi-Fi Direct vůbec nepodporují. Vytvořením nové skupiny se automaticky stává zařízení s podporou Wi-Fi Direct jejím vlastníkem. Tuto skupinu lze vytvořit pomocí metody WifiP2pManager.createGroup(). Všechny ostatní zařízení se mohou připojit do této skupiny jako klienti, pokud znají přístupové heslo a název skupiny. Heslo a název skupiny je náhodně vygenerován pomocí WPS při vytvoření skupiny a zjistit jej může jedině vlastník skupiny pomocí metody getPassphrase() a getNetworkName() z třídy WifiP2pGroup. Nastavit jej nelze, i když existují metody setNetworkName() a setPassphrase(), tak tyto metody jsou neveřejné a nelze k nim přistupovat. Avšak v programu pro tuto diplomovou práci lze jméno skupiny nastavit. Nastavení je dosaženo pomocí využití metody reflexe.

Klienti v Androidu komunikují po připojení pouze s vlastníkem skupiny. Pokud chtějí komunikovat s dalšími klienty ve skupině, je nutné zajistit tuto komunikaci přes vlastníka skupiny, který ji přepošle správnému klientovi. Řešení je tedy možné jen na úrovni aplikace, protože Wi-Fi Direct neobsahuje tuto funkcionalitu. Komunikace mezi klientem a vlastníkem skupiny je šifrována pomocí WPA2 a metody AES s 256 bitovým šifrováním.

4.1.6 **Informace o skupině a spojení**

Po připojení lze získat informace o skupině nebo spojení pomocí rozhraní z třídy WifiP2pManager a registrovaného Broadcast Receiveru. Tento Broadcast Receiver musí obsahovat Intent filtr stejný jako na níže uvedeném kódu.

```
IntentFilter mIntentFilter = new IntentFilter();
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_AC
TION);
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_AC
TION);
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANG
ED_ACTION);
mIntentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHAN
GED_ACTION);
```

Pomocí použití rozhraní WifiP2pManager.ConnectionInfoListener a WifiP2pManager.GroupInfoListener lze získat informace pomocí tohoto Broadcast Receiveru o vytvořené skupině, ale také i tom, které zařízení je vlastníkem skupiny a které klientem. V Broadcast Receiveru lze použit pak tento, níže uvedený, úsek kódu.

```
if
(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action
)) {
    if (mManager != null) {
        NetworkInfo networkInfo =
intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
        if (networkInfo.isConnected()) {
            mManager.requestConnectionInfo(mChannel, mService);
            mManager.requestGroupInfo(mChannel, mService);
        }
    }
```

4.1.7 **Povolení**

Každá aplikace musí pro práci s Wi-Fi Direct obsahovat v Android Manifestu tyto povolení. Bez těchto povolení aplikace nebude fungovat a je nutné je vždy využít.

```
<uses-permission
        android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission
        android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission
        android:name="android.permission.INTERNET"/>
```

Od verze Androidu 6.0 s API 23 došlo k odstranění metod pro přístup k MAC adrese zařízení. Metoda WifiInfo.getMacAddress() nyní vrací konstantní hodnotu 02:00:00:00:00:00.

Pokud zařízení s Androidem 6.0 se pokusí vyhledat zařízení v okolí, tak je tato operace viditelná pro okolní zařízení jako vyhledávaní z náhodné MAC adresy.

Pro přistup k MAC adresám ostatních zařízení v okolí, je nutné, aby aplikace obsahovala dále tyto povolení, které je nutné potvrdit za běhu aplikace.

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

5 Aplikace

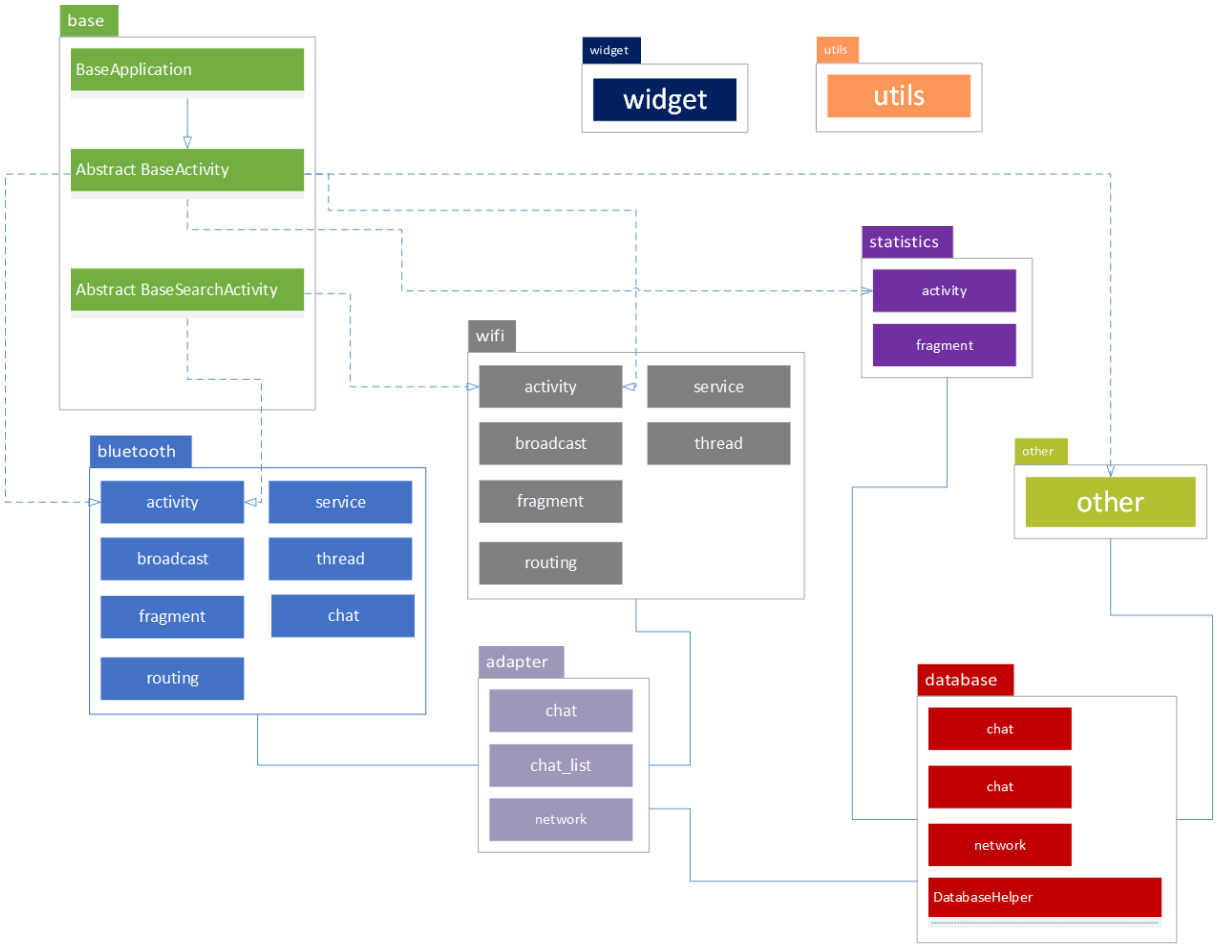
Cílem této diplomové práce bylo také vytvoření ukázkové aplikace, která umožní zabezpečenou a nezabezpečenou komunikaci mezi klienty v sítí navzájem připojených mobilních telefonů pomocí Ad-hoc sítí. Jak je však popsáno v kapitole 2, Android API a jím podporované technologie Bluetooth a Wi-Fi Direct neobsahují podporu Ad-hoc komunikace, která by umožňovala vytvoření sítě, která by obsahovala sobě rovné komunikační uzly. Z tohoto důvodu nešlo zajistit, aby každá aplikace fungovala jako uzel pro ostatní aplikace. Jedinou podporovanou možností je vytvoření serveru, který dokáže pomocí vnitřního přenosového protokolu přeposílat a přijímat zprávy od jeho klientů. Pokud však tento server vypadne, vytvořená síť se rozpadne a je nutné, aby se jedno ze zařízení stalo novým serverem a všichni klienti se k němu znovu připojili. Aplikace také umožňuje vyhledání Bluetooth zařízení v okolí a možnost připojení k těmto zařízením. Pro Wi-Fi Direct aplikace umožňuje vyhledání všech zařízení s aplikací v okolí. K nalezených zařízením se pak lze zabezpečeně nebo nezabezpečeně připojit.

Aplikace byla vyvíjena v programu Android Studio v jeho poslední a aktuální verzi 2.1 s verzí jazyka Gradle 2.1. Funkční část aplikace je napsána v jazyce Java a její vzhled v jazyce XML. Testování a kompilace probíhala z velké části pouze na telefonu LG Nexus 4 s nainstalovaným Androidem verze 5.1.1, který byl následně aktualizován na verzi 6.0.1 s aktuálním zabezpečením k dubnu 2016. Dále byla aplikace zkoušena se zapůjčeným tabletem Prestigio MultiPad 7.0 s Androidem 4.1. Všechny podpůrné knihovny byly použity v poslední verzi 23.3.0. Jako minimální podporovanou verzi Androidu jsem si stanovil verzi s API 16, tedy Android 4.1. Důvodem je, že samotný Android podporuje určité funkce Wi-Fi Direct a Bluetooth právě až od této verze. Kompatibilitu s nižšími verzemi tak nelze zajistit. Kompilační nástroje jsem využil v poslední stabilní verzi 23.3.0. Maximální podporovaná verze API a verze, ke které byl program kompilován, je API 23, tedy Android 6.0.

5.1 Architektura

Architektura aplikace je tvořena devíti částmi. Každá část obsahuje Java kód s funkcemi, na které se odkazuje název dané části. Na níže uvedeném třídním diagramu je zakresleno funkční propojení jednotlivých částí mezi sebou. V další kapitole popíšu všechny jednodušší části aplikace, jejich účel v aplikaci a funkce, které obsahují. V následujících kapitolách se pak budu věnovat hlavním částem této diplomové práce, které se věnují Bluetooth a Wi-Fi Direct. Součástí popisu těchto dvou hlavních částí bude rovněž popis Services a AIDL, které do každé části spadají.

Program dále obsahuje rozhraní psané v jazyce XML. Toto rozhraní bylo vytvořeno dle pravidel Material Designu [8] od společnosti Google a obsahuje pět hlavních obrazovek. Popisem vzhledu a rozhraní aplikace se budu zabývat v poslední kapitole, kde rovněž popíšu všechny využité knihovny při implementaci aplikace.



Obrázek 1.10: *Architektura aplikace*

5.2 Části aplikace

5.2.1 Část Base

Tato část je hlavní částí programu. Obsahuje třídy, které obsahují funkce, které jsou společné pro celou aplikaci. Hlavní třídou je třída BaseApplication, která je potomkem třídy Application. Tato třída má na starosti správu služeb aplikace a rovněž kontrolu konzistence aplikace v paměti mobilního zařízení.

Další třídou je abstraktní BaseActivity, která je hlavní třídou pro všechny Activity. Všechny Activity v aplikaci jsou tedy jejími potomky. Tato třída zahrnuje základní funkčnost pro všechny Activity. Díky této třídě lze zobrazit navigační menu NavigationView z každé Activity, ale rovněž se stará o ukládání nastavení pro aplikaci nebo získává BluetoothAdapter a WifiManager pro celou aplikaci. Dále tato BaseActivity spouští při prvním spuštění aplikace Service třídu WifiService. Stará se rovněž o kontrolu, zda je tato třída spuštěna či nikoliv.

Abstraktní třída BaseSearchActivity se stará o správu vyhledávání pro každou Activity. Tato třída obsahuje implementaci knihovny SearchView [6]. Vyhledávání probíhá pouze v uložených zařízeních v databázi aplikace a to v tabulkách CHAT_LIST_TABLE.

A nakonec poslední třída v této části je implementace Broadcast Receiver třídy BaseSwitchBroadcastReceiver, která se stará o správu přepínačů pro zapínání a vypínaní Bluetooth a Wi-Fi v aplikaci.

5.2.2 Část Statistics

Část Statistics obsahuje jednu třídu StatisticsActivity a dvě třídy StatisticsFragmentBt a StatisticsFragmentWifi. Třída StatisticsActivity je základní třídou pro zobrazení obou Fragmentů. Umožňuje přístup k vyhledávání v aplikaci a přístup k zobrazení obsahu obou Fragmentu.

Oba Fragmenty obsahují možnost vypínání a zapínání Bluetooth a Wi-Fi pomocí přepínače, který je ovládán pomocí Broadcast Receiveru. Dále tyto Fragmenty zobrazují statistiku používání aplikace pomocí grafu MarkView z knihovny Design, který zobrazuje počet zpráv v databázi pro Bluetooth a Wi-Fi.

5.2.3 Část Other

V této části jsou další třídy Activity, jako například vyhledávací třída SearchActivity, jenž zobrazuje výsledky vyhledávání v uložených zařízeních. Dále je zde třída AboutActivity, která zahrnuje informace o aplikaci a použitých knihovnách. Poslední třída SettingsActivity slouží pro nastavení aplikace. Lze zde vymazat všechny tabulky databáze najednou nebo nastavit si tmavé zobrazení uživatelského rozhraní pro práci s aplikací v noci.

5.2.4 Část Adapter

Část Adapter obsahuje několik tříd. Třída Cab slouží pro vyvolání menu nad seznamem na obrazovce s nalezenými nebo uloženými Bluetooth a Wi-Fi Direct zařízeními. Toho lze dosáhnout pomocí dlouhého kliknutí na zobrazenou položku. Třídy Chat a ChatList slouží jako datové typy pro třídy ChatAdapter a ChatListAdapter a jsou nutné pro zobrazení seznamu zařízení pomocí seznamu RecyclerView. Třídy Chat, ChatList a Network zároveň slouží jako datové typy pro databázi. Poslední třídou je třída ViewPagerAdapter, která slouží pro práci s Fragmenty.

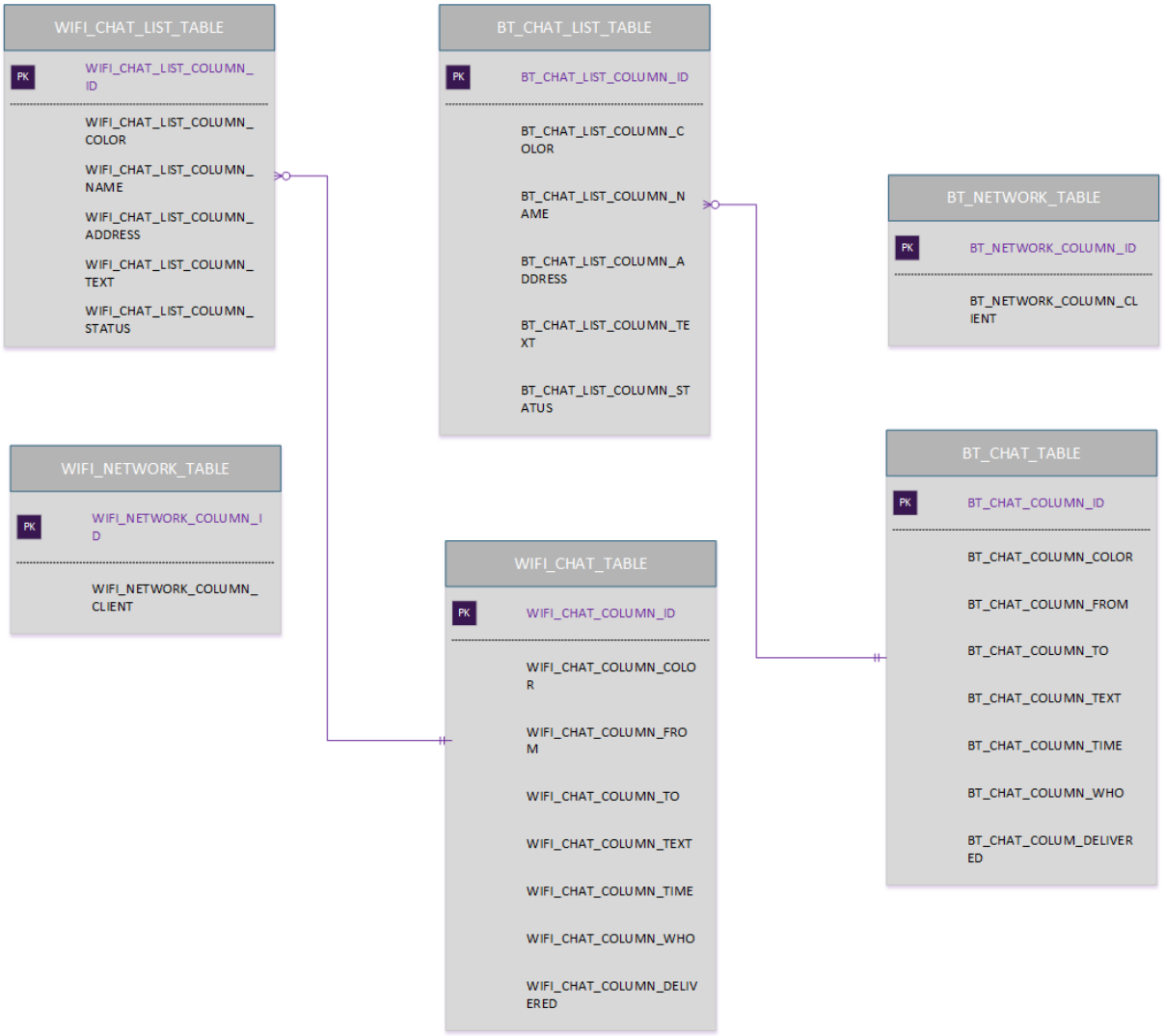
5.2.5 Část Widget

Část Widget obsahuje implementaci tříd, které se starají o prvky uživatelské rozhraní. Třída ScrollSwipeRefreshLayout řeší podporu AppBarLayoutu a slouží pro spuštění vyhledávání Bluetooth a Wi-Fi Direct zařízení. Třída CustomItemAnimator se stará o animaci nových položek v seznamu zařízení.

5.2.6 Část Utils

Tato část obsahuje čtyři třídy, které obsahují společné statické metody a proměnné pro celou aplikaci. Třída Codes obsahuje statické proměnné, které se používají pro zasílání zpráv v aplikaci, ale také obsahuje unikátní Bluetooth UUID kódy pro aplikaci. Třída Intents obsahuje statické proměnné, které slouží pro směrování a přeposílaní zpráv v aplikaci. Třída Methods obsahuje statické metody, které jsou využívány na více místech v kódu aplikace. Například obsahuje Bluetooth metodu ensureDiscoverable(), která povoluje zviditelnění zařízení pro ostatní nespárovaná Bluetooth zařízení. Dále obsahuje třeba metodu getDeviceStatus(), která vrací status zařízení pro Bluetooth a Wi-Fi Direct a další metody. Třída Notifications se stará o upozornění, která se zobrazují po přijetí zprávy, a to jak pro Bluetooth, tak i pro Wi-Fi Direct. Třída AES obsahuje metody, které slouží k šifrování příchozích a odchozích zpráv pomocí šifrování AES.

5.2.7 Část Database



Obrázek 1.11: *Schéma databáze*

Tato část obsahuje implementaci SQLite databáze. Obsahuje třídu DatabaseHelper, která obsahuje Java metody a SQLite dotazy pro vytvoření, aktualizaci a mazání všech tabulek. Další třída DatabaseDeleteAll obsahuje metodu s transakcí na smazání všech tabulek najednou.

Třídy v částech chat, chat_list a network jsou implementací každé jednotlivé tabulky ve verzi pro Bluetooth a Wi-Fi. Každá tato třída obsahuje metody pro práci s tabulkou a obsahuje kontrolu, jestli daný záznam již v tabulce existuje. Struktura tabulek pro Wi-Fi a Bluetooth je stejná, ale z důvodu oddělení má každá technologie svou tabulku.

Tabulky CHAT_LIST_TABLE ukládají každé zařízení, ke kterému se zařízení připojí. Primárním klíčem je zde ID zařízení. Tabulky CHAT_TABLE ukládají historii zpráv mezi zařízeními. Primárním klíčem je zde ID zprávy a cizím klíčem ID zařízení z tabulky CHAT_LIST. Tabulky NETWORK_TABLE ukládají klientská zařízení a primárním klíčem je zde opět ID zařízení.

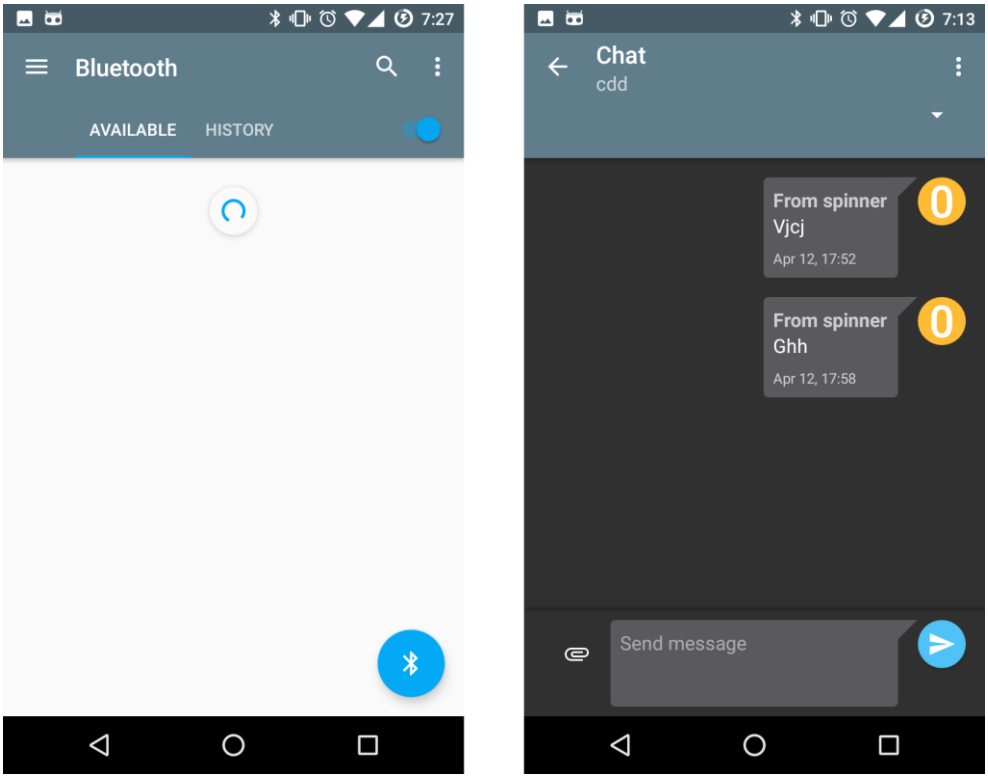
5.3 Část Bluetooth

Implementace klasického Bluetooth v aplikaci obsahuje možnost vyhledání nových a spárovaných zařízení, vytvoření serveru a připojení klienta k tomuto serveru. Dále umožňuje zabezpečenou a nezabezpečenou RFCOMM komunikaci mezi serverem a připojenými klienty včetně přeposílaní zpráv mezi klienty prostřednictvím serveru za pomoci implementovaného protokolu. Tento protokol potřebuje další vývoj a testování.

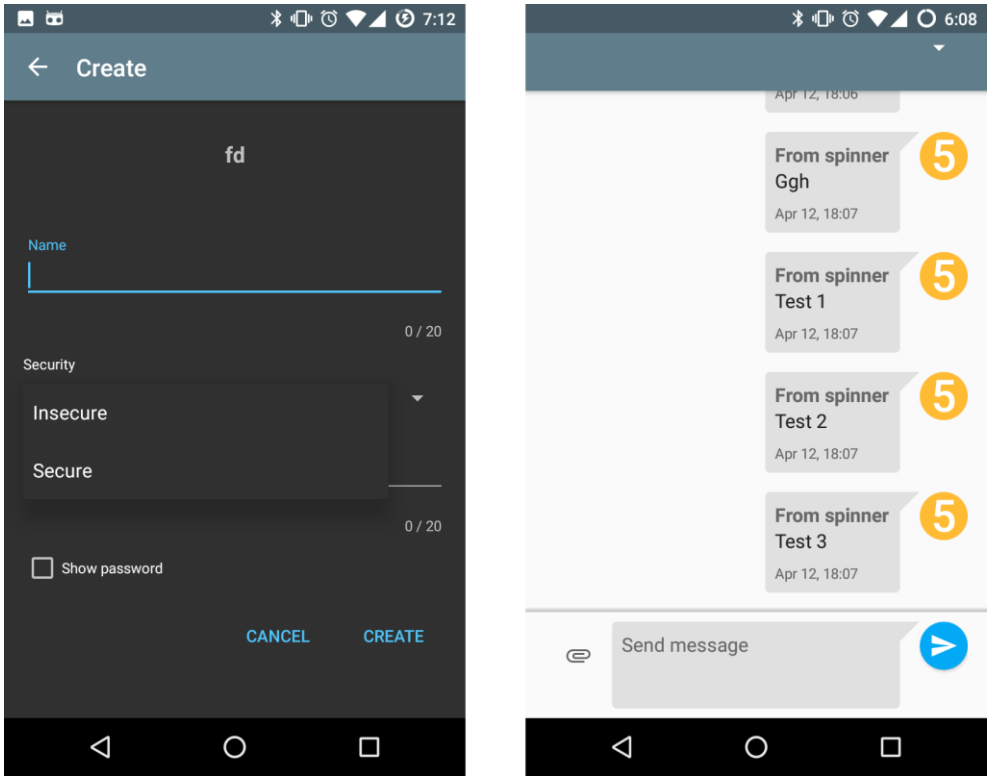
5.3.1 Klient a server

Klientská část je reprezentována třídami BtActivity a BtChatClientActivity. Třída BtActivity obsahuje dva Fragmenty BtAvailableFragment a BtSavedFragment. BtSavedFragment je seznam, který tvoří historie všech připojených zařízení. Tento seznam je tvořen záznamy z tabulky BT_CHAT_LIST_TABLE. Ve třídě BtAvailableFragment lze vyhledat všechna nová nebo již spárovaná zařízení a přidat je do seznamu na obrazovce. Po výběru položky ze seznamu se spustí třída BtChatClientActivity, ve které je možné se připojit pomocí zabezpečeného nebo nezabezpečeného připojení k tomuto zařízení a následně posílat textové zprávy. Toto zabezpečené a nezabezpečené připojení k zařízení je popsáno v kapitole 3.2.4. Podmínkou následné komunikace je, aby toto zařízení obsahovalo stejnou aplikaci se stejným UUID. Zároveň musí mít spuštěnou Activity třídu BtChatServerActivity se spuštěnou Service třídou BtServiceServer.

Serverová část je tvořena třídami BtCreateActivity a BtChatServerActivity. Přistup k BtCreateActivity je možný pomocí plovoucího tlačítka v BtActivity. Tato třída slouží k vytváření a úpravě nastavení serveru. Lze zvolit možnost zabezpečeného nebo nezabezpečeného serveru (viz. kapitola 3.2.5) a také lze nastavit jeho jméno. Po zadání těchto údajů dojde k spuštění BtChatServerActivity a lze komunikovat s připojenými klienty.



Obrázek 1.12: *BtActivity a BtChatClientActivity*



Obrázek 1.13: *BtCreateActivity a BtChatServerActivity*

5.3.2 Services

Jak bylo uvedeno v předchozích kapitolách, Android obsahuje několik stavů Activit. Pokud uživatel minimalizuje aplikaci a rozhodne se pracovat s jinou, došlo by tak k přerušení možnosti dostávat zprávy. Proto aplikace využívá služby Service běžící na pozadí. Tyto služby se spouští ve BtChatClientActivity a BtChatServerActivity a využívají dvou rozhraní ze souborů AIDL. Služby běží od spuštění Activity na pozadí, dokud se je uživatel nerozhodne v aplikaci ukončit. Pro komunikaci Activity se službou, je to IBtService a pro komunikaci služby s Activity, je to IBtMessengerCallback. Tyto rozhraní pak implementují tři třídy Service. Třída BtServiceClient běží na pozadí a umožňuje komunikaci mezi klientem a serverem. Třída BtServiceServer umožňuje serveru komunikovat s klienty. A nakonec třída BtService slouží jako základová třída pro obě předchozí třídy. Tyto Services využívají implementaci komunikačního protokolu ve třídě BtProtocol.

Pro připojení a zabezpečenou a nezabezpečenou komunikaci serveru s klienty běží ve všech Service třídách dvě vlákna BtAcceptThread a BtConnectedThread.

5.3.3 Komunikace mezi klienty

Android a Bluetooth neumožňuje přeposílání zpráv mezi klienty, pouze mezi klientem a serverem. Implementovaný komunikační protokol pro síť Piconet se nachází ve třídě BtProtocol a umožnuje přeposílaní zprávy od jednoho klienta k serveru nebo k jinému klientovi přes server. Je nutné uvést, že protokol je třeba dokončit a upravit jeho funkčnost, jelikož při vývoji nebylo možné testovat na více než na jediném a dočasně na dvou zařízeních zároveň. Zároveň dochází k občasným problémům při spojení mezi různými verzemi Androidu. Důvodem těchto problému jsou pravděpodobně značné změny Bluetooth Android API mezi jednotlivými verzemi operačního systému a také výměna Bluetooth stacku v Androidu 4.2. Na testovacím zařízení LG Nexus 4 s Androidem 6.0.1, i přes využití povolení (více v kapitole 3.2.6), tato komunikace nefunguje, jak by měla.

Zprávy mezi zařízeními se posílají pomocí zabezpečeného nebo nezabezpečeného spojení a instance třídy Intent s parametrem Intents.BT_INTENT_CHAT_CLIENT_MESSAGE pro klienta a pro server Intents.BT_INTENT_CHAT_SERVER_MESSAGE. Každá posílaná zpráva obsahuje základní čtyři údaje, které jsou šifrovány pomocí AES šifrování ze třídy AES. Od koho je daná zpráva, komu je určena, text zprávy a čas jejího odeslání.

```
Intent intent = new Intent(Intents.BT_INTENT_CHAT_MESSAGE);
        intent.putExtra("from", from);
        intent.putExtra("to", to);
        intent.putExtra("text", text);
        intent.putExtra("time", time);
// intent.putStringArrayListExtra("clients", clients);
mBoundService.sendMessage(Codes.BROADCAST, intent);
```

Každá zašifrovaná zpráva je poslána pomocí metody sendMessage(). Metoda má dva parametry. První parametr String Address určuje, komu je zpráva určena a druhý parametr obsahuje samotnou instanci třídy Intent s obsahem zprávy. Oba parametry jsou v této metodě převedeny na vlastní datový typ ByteMessage a poslány do metody sendByteMessage().

V metodě sendByteMessage() se, dle String parametru Adress získaného ze zprávy, pozná, jaké je směrování zprávy. Pokud se rovná Codes.DIRECT_MESSAGE, jedná se o přímou zprávu pro zařízení. Pokud je tato adresa jiná, dojde k jejímu přeposlání.

Metoda receiveMessage() se stará o přijmutí a rozšifrování zprávy nebo její případné přeposlání. Opět se zde využívá kontrola parametru Address. Pokud se rovná Codes.DIRECT_MESSAGE, zpráva je přijata cílovým zařízením a poslána do metody processIntent(). Pokud je adresa jiná, dojde opět k jejímu přeposlání.

Metoda processIntent() se stará o ukládání každé přijaté zprávy do databáze, upozornění na přijatou zprávu a její zobrazení na cílovém zařízení. K zobrazení zprávy v Activity se pak využívá instance IBtMessengerCallback. V metodě processIntent() se pomocí tří Intentů rozlišuje určení přijaté zprávy a také, kdo ji odeslal. Pro Intents.BT_INTENT_CHAT_SERVER_MESSAGE platí, že zpráva pochází od serveru a je určena pro klienta. Každý klient ze zprávy od serveru získává seznam všech zařízení, který ukládá do databázové tabulky BT_NETWORK_TABLE. Intents.BT_INTENT_CHAT_CLIENT_MESSAGE je pak zprávou klienta pro server.

Každá úspěšně doručená zpráva posílá na svého odesilatele potvrzení s Intents.BT_INTENT_CHAT_DELIVERED_MESSAGE. Toto potvrzení zobrazí TOAST zprávu s textem „ Zpráva přes Bluetooth byla doručena". Tímto je pak potvrzeno, že zpráva byla doručena. Pokud zpráva nedojde, je nadále uložena v zařízení odesilatele, kde čeká na odeslání.

```
Intent deliveredMessageIntent = new
Intent(Intents.BT_INTENT_CHAT_DELIVERED_MESSAGE);

deliveredMessageIntent.putExtra("to", to);

deliveredMessageIntent.putExtra("delivered",mContext.getString(R.str
ing.delivered_message_bt));

sendMessage(Codes.DIRECT_MESSAGE, deliveredMessageIntent);
```

5.3.4 Notifikace a Broadcast Receivery

Jelikož aplikace pro komunikaci využívá Services, tak je možné aplikaci minimalizovat a stále lze dostávat zprávy, dokud služba Service bude běžet na pozadí. Přenosový protokol ve třídě BtProtocol umí poznat, kdy je aplikace spuštěná a kdy je na pozadí. V případě běhu aplikace na pozadí dokáže zobrazit upozornění, neboli notifikaci, na přijatou zprávu. Pro upozornění na zprávu se využívá metod ze třídy Notifications z části Utils.

Třída BtSwitchBroadcastReceiver se stará o nastavení přepínače Switch pro zapnutí a vypnutí Bluetooth.

Každý server obsahuje v BtServiceServer registrovaný Broadcast Receiver. Pomocí toho Broadcast Receiveru ve třídě BtServiceServerBroadcastReceiver si server ukládá seznam BT_ADDR všech svých připojených klientů a následně tento seznam ukládá také do databázové tabulky BT_NETWORK_TABLE. Ukládání do seznamu lze vidět na níže uvedeném kódu.

```
private final Set<BluetoothDevice> connectedDevices = new
HashSet<>();
private List<String> connectedDevicesString = new ArrayList<>();
```

```
if (BluetoothDevice.ACTION_ACL_CONNECTED.equals(action)) {
Toast.makeText(context, "We are now connected to " +
device.getName(), Toast.LENGTH_SHORT).show();
```

```
if (!mConnectedDevices.contains(device))
     mConnectedDevices.add(device);
if (!mConnectedDevicesString.contains(device.getAddress()))
     mConnectedDevicesString.add(device.getAddress());}
```

Seznam zařízení následně získává z BtService třída BtChatServerActivity pomocí metody getListOfClients(), ve které se tento seznam zobrazuje jako položky menu ve Spinneru. Server tento seznam všech klientů posílá v každé zprávě všem klientům jako pátý údaj v instanci třídy Intent, která obsahuje zprávu. Klienti si pak tento seznam ukládají do tabulky BT_NETWORK_TABLE a následně jej zobrazují ve Spinneru v BtChatClientActivity. Zde pak slouží k určení parametru String Address pro metodu sendMessage() z třídy BtProtocol.

Klientská Service třída BtServiceClient také obsahuje registrovaný Broadcast Receiver. Třída BtServiceClientBroadcastReceiver reaguje pomocí Intentu BluetoothDevice.ACTION_ACL_DISCONNECTED na ukončení spojení klienta se serverem. Pokud se toto spojení ukončí a klient vypadne, automaticky se čtyřikrát pokusí znovu připojit k serveru a pokračovat opětovně v komunikaci. BT_ADDR serveru si klient ukládá do nastavení SharedPreferences při každém spojení se serverem, takže je schopen se k serveru znovu připojit pomocí této adresy.

```
if (BluetoothDevice.ACTION_ACL_DISCONNECTED.equals(action)) {
mProtocol.connect(BT_ADDR ze SharedPreferences);
Toast.makeText(mContext, "Connecting to Server ...",
Toast.LENGTH_SHORT).show();}
```

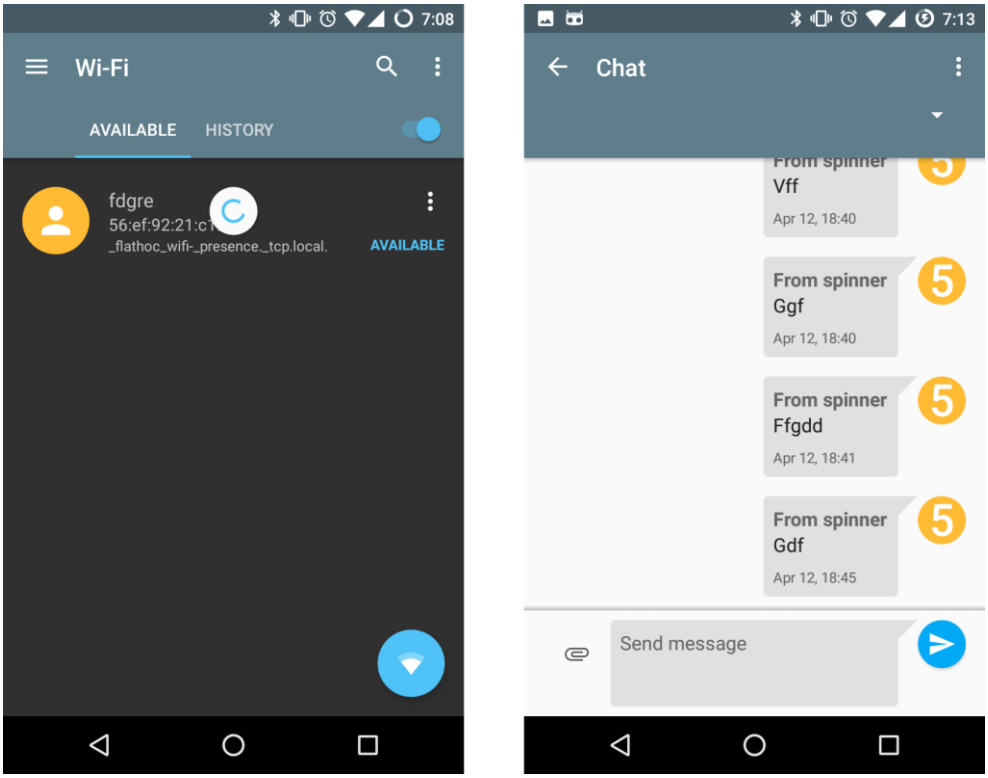
5.4 Část Wifi

Implementace Wi-Fi společně s implementací Wi-Fi Direct P2P služby v aplikaci obsahuje možnost vyhledání zařízení, které obsahují aplikaci, vytvoření skupiny a připojení klienta k tomuto vlastníkovi skupiny. Dále umožňuje komunikaci mezi vlastníkem skupiny a připojenými klienty včetně přeposílaní zpráv mezi klienty prostřednictvím vlastníka skupiny za pomoci implementovaného protokolu.

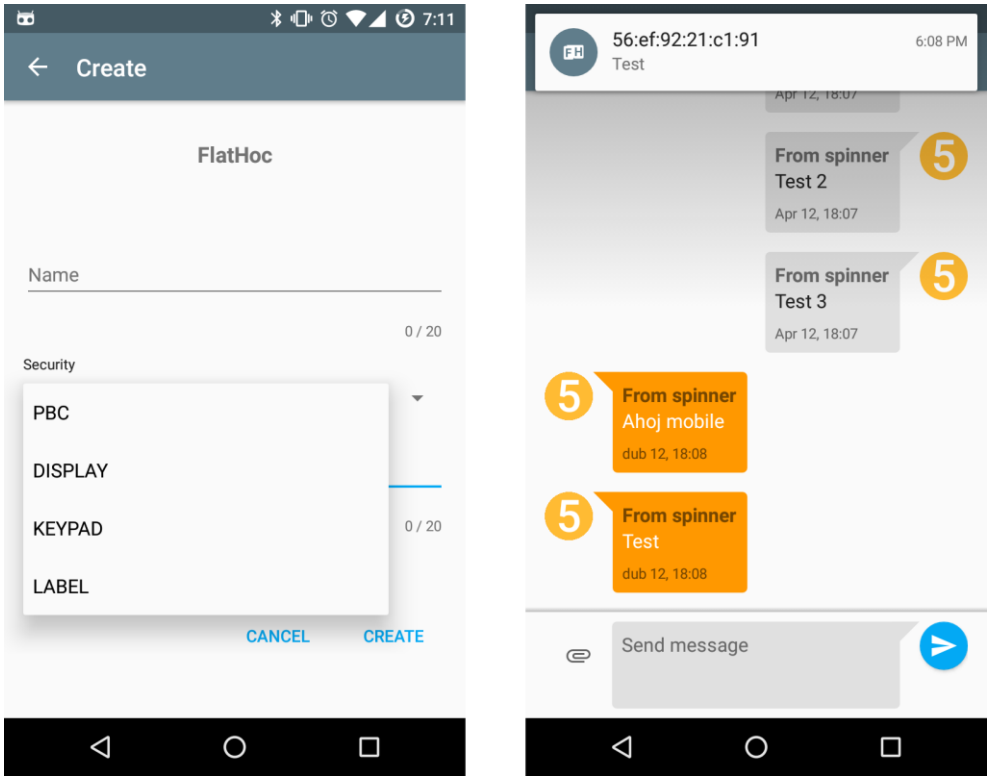
5.4.1 Klient a vlastník skupiny

Klientská část je tvořena třídami WifiActivity a WifiChatClientActivity. Třída WifiActivity stejně jako v Bluetooth části obsahuje dva Fragmenty. Fragment WifiSavedFragment zobrazuje historii všech připojených zařízení. Tyto údaje získává z tabulky WIFI_CHAT_LIST_TABLE. Fragment WifiAvailableFragment umožňuje hledání Wi-Fi Direct zařízení se stejnou spuštěnou aplikací s P2P službou běžící nad Wi-Fi Direct. Všechna vyhledaná zařízení se zobrazují jako nové položky v seznamu. Výběrem položky z tohoto seznamu se spustí třída WifiChatClientActivity. V této Activity třídě se lze připojit čtyřmi způsoby k zařízení. Lze se připojit pomocí volby PBC, DISPLAY, KEYPAD a LABEL, které jsou popsány v kapitole 4.1.4. Po připojení je pak možné komunikovat se zařízením a posílat si textové zprávy. Podmínkou připojení a komunikace je, aby obě zařízení měly spuštěnou aplikaci se službou Wi-Fi Direct a musí na nich běžet Service třída WifiService.

Serverová část je tvořena třídami WifiCreateActivity a WifiChatGroupOwnerActivity. Třídu WifiCreateActivity lze spustit plovoucím tlačítkem v pravém dolním rohu. Tato třída slouží k vytváření a úpravě nastavení skupiny. Lze zvolit, jaká možnost zabezpečení bude povolena (viz. kapitola 4.1.4) a lze nastavit jméno vlastníka skupiny. Po vložení těchto údajů se vytvoří skupina, zařízení se stane jejím vlastníkem a lze komunikovat s klienty. Obrázky níže obsahují obě Activity.



Obrázek 1.14: *WifiActivity a WifiChatClientActivity*



Obrázek 1.15: *WifiCreateActivity a WifiChatGroupOwnerActivity*

5.4.2 Services

Jak již jsem popisoval v části Bluetooth, tak Android obsahuje několik stavů Activit. Pokud uživatel minimalizuje aplikaci a rozhodne se pracovat s jinou, došlo by k přerušení možnosti dostávat zprávy. Proto tato část aplikace využívá opět služby Service běžící na pozadí. Tyto služby se spouští ve WifiChatClientActivity a WifiChatGroupOwnerActivity a využívají dvou rozhraní ze souborů AIDL. Služby běží od spuštění aplikace na pozadí, dokud se je uživatel nerozhodne v aplikaci ukončit. Pro komunikaci Activity se službou je to IWifiService a pro komunikaci služby s Activity, je to IWifiMessengerCallback. Tyto rozhraní pak implementují Service třída WifiServiceClient a její rodičovská třída WifiService. Obě třídy běží na pozadí a umožňují komunikaci ve skupině pomocí implementace komunikačního protokolu ve třídě WifiProtocol.

V každé Service třídě pak dále běží dvě vlákna. Vlákno WifiClientThread slouží k připojení klienta k vlastníkovi skupiny. Vlákno WifiGroupOwnerThread slouží k připojení klientů k vlastníkovi skupiny.

5.4.3 Komunikace mezi klienty

Implementace Wi-Fi Direct v Androidu a obecně neumožňuje přeposílaní zpráv mezi klienty, pouze mezi vlastníkem skupiny a klientem. Implementovaný komunikační protokol pro Wi-Fi Direct skupinu se nachází ve třídě WifiProtocol a umožňuje přeposílaní zprávy od klienta k vlastníkovi skupiny nebo od klienta ke klientovi přes vlastníka skupiny. Tento protokol je však nutné dokončit a upravit jeho funkčnost, jelikož jeho vývoj byl možný pouze na dvou zařízeních, což neumožňovalo kvalitní testování. Wi-Fi Direct API je velmi chybové a občas zařízení nejsou vzájemně nalezena, i když jsou tato dvě zařízení hned vedle sebe. Díky chybovosti a nestabilitě Wi-Fi Direct není mnoho aplikací v obchodě Google Play, které by toto API využívaly.

Zprávy mezi zařízeními se posílají pomocí instance třídy Intent a parametru Intents.WIFI_INTENT_CHAT_CLIENT_MESSAGE pro klienta a pro vlastníka skupiny pomocí parametru Intents.WIFI_INTENT_CHAT_GROUP_OWNER_MESSAGE. Každá posílaná zpráva obsahuje stejně jako ve Bluetooth části čtyři údaje, které jsou šifrovány pomocí AES šifrování z třídy AES. Prvním údajem je, od koho je daná zpráva. Druhým je, komu je zpráva určena. Třetím je text zprávy a posledním čtvrtým údajem je čas odeslání zprávy.

```
Intent intent = new Intent(Intents.BT_INTENT_CHAT_MESSAGE);
        intent.putExtra("from", from);
        intent.putExtra("to", to);
        intent.putExtra("text", text);
        intent.putExtra("time", time);
// intent.putStringArrayListExtra("clients", clients);
mBoundService.sendMessage(Codes.BROADCAST, intent);
```

Každá zpráva je před odesláním zašifrována. Pro odeslání zprávy slouží metoda sendMessage(). Tato metoda obsahuje dva vstupní parametry. První parametr String Address slouží pro určení cíle zprávy. Druhy parametr obsahuje instanci třídy Intent s obsahem zprávy. Tyto dva parametry se v této metodě převádí na vlastní datový typ ByteMessage a následně jsou poslány do metody sendByteMessage().

V metodě sendByteMessage() se podle hodnoty parametru String Address získaného ze zprávy pozná, jaký je cíl zprávy. Pokud se hodnota rovná Codes.DIRECT_MESSAGE, jedná se o přímou zprávu pro cílové zařízení. Pokud je tato hodnota jiná, dojde k jejímu přeposlání.

Pro přijmutí a rozšifrování zprávy je na cílovém zařízení metoda receiveMessage(). Tato metoda opět využívá kontroly parametru String Address. Pokud se hodnota rovná Codes.DIRECT_MESSAGE, zpráva je přijata cílovým zařízením a poslána do metody processIntent(). Pokud je hodnota jiná, dojde k přeposlání zprávy.

Metoda processIntent() zpracovává každou přijatou zprávu. Stará se o její ukládání do databáze, o zobrazení upozornění na přijatou zprávu a také o její zobrazení v Activity pomocí instance IWifiMessengerCallback. Tato metoda rozlišuje pomocí tří Intentů určení přijaté zprávy a jejího odesilatele. Intent Intents.WIFI_INTENT_CHAT_GROUP_OWNER_MESSAGE je určením, že zpráva pochází od vlastníka skupiny a je určena pro klienta. Každý klient získává od vlastníka skupiny seznam zařízení, které si ukládá do databázové tabulky WIFI_NETWORK_TABLE.

Intent Intent.WIFI_INTENT_CHAT_CLIENT_MESSAGE je určením, že zpráva pochází od klienta a je určena pro vlastníka skupiny. Pokud je zpráva úspěšně doručena, dojde k poslání potvrzovací zprávy odesilateli. Tato zpráva obsahuje Intents.WIFI_INTENT_CHAT_DELIVERED_MESSAGE a zobrazí odesilateli TOAST zprávu s textem „ Zpráva přes Wi-Fi Direct byla doručena". Tímto je pak potvrzeno, že zpráva byla doručena. Pokud zpráva nedojde, je nadále uložena v zařízení odesilatele, kde čeká na odeslání.

```
Intent deliveredMessageIntent = new
Intent(Intents.WIFI_INTENT_CHAT_DELIVERED_MESSAGE);deliveredMessageI
ntent.putExtra("to", to);
deliveredMessageIntent.putExtra("delivered",mContext.getString(R.str
ing.delivered_message_wifi)); sendMessage(Codes.DIRECT_MESSAGE,
deliveredMessageIntent);
```

5.4.4 Notifikace a Broadcast Receivery

Stejně jako v Bluetooth části aplikace, tak také Wi-Fi Direct část aplikace pro komunikaci využívá Services. Lze tedy aplikaci minimalizovat a stále dostávat zprávy, dokud služba Service bude běžet na pozadí. Přenosový protokol ve třídě WifiProtocol umí poznat, kdy je aplikace spuštěná a kdy je na pozadí. V případě běhu aplikace na pozadí dokáže zobrazit upozornění, neboli notifikaci, na přijatou zprávu. Pro upozornění na zprávu se využívá metod ze třídy Notifications z části Utils.

Aplikace v části Wifi obsahuje implementaci Broadcast Receiveru s názvem WifiSwitchBroadcastReceiver, který se stará o vypínání a zapínaní Wi-Fi pomocí přepínače.

Každý vlastník skupiny si ukládá seznam všech MAC adres zařízení do tabulky WIFI_NETWORK_TABLE v databázi. Ukládání seznamu klientů u vlastníka skupiny slouží metoda groupInfo() ve třídě WifiProtocol. Ta získává svůj parametr WifiP2pGroup z rozhraní WifiP2pManager.onGroupInfoAvailable(), které je implementováno ve WifiService třídě a využívá Broadcast Receiver ve třídě WifiServiceServerBroadcastReceiver.

```
WifiNetworkTable db = new WifiNetworkTable(mContext);
Collection<WifiP2pDevice> devicesInGroup = new ArrayList<>();
devicesInGroup.addAll(group.getClientList());
for (WifiP2pDevice address : devicesInGroup) {
     db.addItem(new Network(address.deviceAddress));}
Methods.getMessage(mContext, group.getInterface() + " " +
group.getNetworkName() + " " + group.getPassphrase());
```

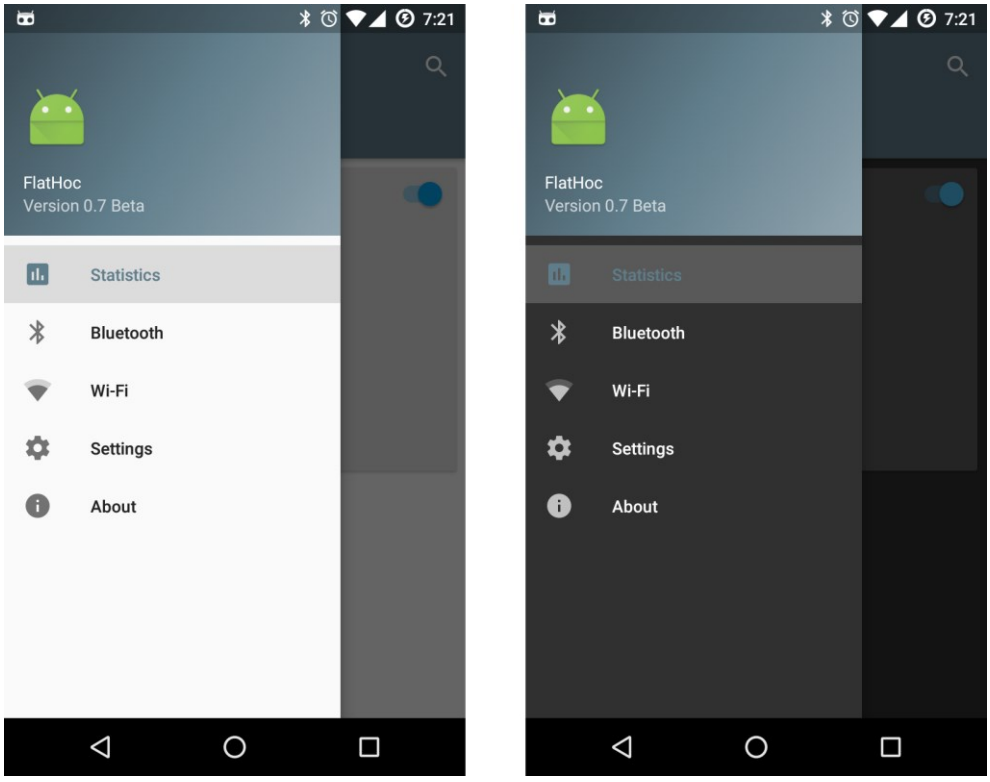
Seznam zařízení se následně získává z tabulky WIFI_NETWORK_TABLE ve třídě WifiChatGroupOwnerActivity pomocí metody getListOfClients(). V této třídě se tento seznam pak zobrazuje jako položky v menu ve Spinneru. Vlastník skupiny tento seznam klientů posílá v každé zprávě všem svým klientům jako pátý údaj v instanci třídy Intent se zprávou. Klienti si pak tento seznam ukládají do tabulky WIFI_NETWORK_TABLE a následně jej zobrazují ve WifiChatClientActivity jako položky v menu Spinner. Tyto položky v menu Spinner slouží v obou Activity jako změna parametru String Address pro metodu sendMessage() ve třídě WifiProtocol.

Service třída WifiServiceClient pro klienta obsahuje registrovaný Broadcast Receiver ze třídy WifiServiceClientBroadcastReceiver. Tento Broadcast Receiver reaguje na událost WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION a hlídá si, jestli došlo k odpojení klienta od vlastníka skupiny. Pokud je toto spojení ukončeno a klient je odpojen, automaticky se tento klient pokusí čtyřikrát připojit znovu k vlastníkovi skupiny, aby mohl dále pokračovat v komunikaci. MAC adresu vlastníka skupiny si každý klient ukládá do nastavení SharedPreferences při každém spojení s vlastníkem skupiny, takže je klient kdykoliv schopen tuto MAC adresu využít pro opětovné připojení k vlastníkovi skupiny.

```
if
(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {
    if (mManager != null) {NetworkInfo networkInfo =
intent.getParcelableExtra(WifiP2pManager.EXTRA_NETWORK_INFO);
        NetworkInfo.State state = networkInfo.getState();
        if (state == NetworkInfo.State.DISCONNECTED) {
            for (int i = 0; i < 4; i++) {
mProtocol.connect(MAC ze SharedPreferences);
Toast.makeText(mContext, "Connecting to group owner " + (i + 1),
Toast.LENGTH_SHORT).show();
```

5.5 Uživatelské rozhraní a vzhled

Aplikace obsahuje pět hlavních obrazovek (tříd Activity), kdy každá obrazovka odpovídá jedné položce v levém menu NavigationView. Toto vysouvací navigačnímu menu umožňuje přepnutí jednotlivých obrazovek. Každá obrazovka má vytvořené své rozhraní pomocí jazyka XML. Toto rozhraní bylo vytvořeno dle Google Material Designu [8] a v aplikaci existuje rozhraní v tmavé a světlé verzi, jak lze vidět na screenshotech níže. Při použití tmavé verze dochází k nižší spotřebě energie displejem a je rovněž umožněna pohodlnější práce s aplikaci v nočních hodinách. Rozhraní programu je v českém a anglickém jazyce. Soubory strings.xml a strings.xml (cs) obsahují všechny textové řetězce použité v programu. Na následujících obrázcích lze vidět navigační menu NavigationView. První položka v menu odpovídá StatisticsActivity, druhá BtActivity, třetí WifiActivity, čtvrtá SettingsActivity a poslední AboutActivity.



Obrázek 1.16: *Navigační menu aplikace*

5.5.1 Využité knihovny

Pro vytvoření uživatelského rozhraní jsem v diplomové práci využil kromě samotného API z Android SDK také tyto níže uvedené podpůrné knihovny Google s verzí API 23.3.0. Tyto knihovny lze stáhnout přímo v Android Studiu. Využil jsem řadu těchto knihoven, abych přinesl uživateli aplikace softwarové funkce, které nejsou závislé na verzi Androidu, kterou má nainstalovanou v telefonu. Níže uvádím seznam knihoven, které jsem použil při implementaci.

- com.android.support:appcompat-v7:23.3.0
- com.android.support:design:23.3.0
- com.android.support:recyclerview-v7: 23.3.0
- com.android.support:cardview-v7: 23.3.0
- com.android.support:preference-v7: 23.3.0

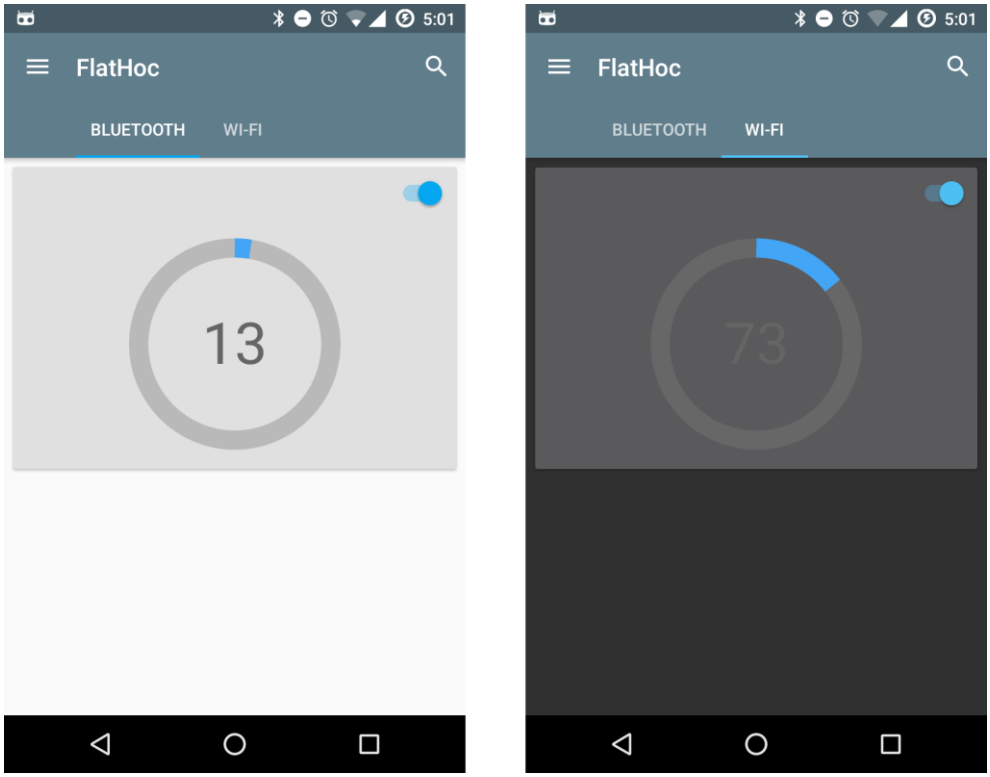
Knihovna AppCompat obsahuje velké množství tříd, které přinášejí vzhled a funkce nových verzí Androidu na starší verze API. Příkladem může byt AppCompatActivity, jenž je variantou Activity, která umožnuje využití nového Material Designu a funkcí Androidu 5.0+ až do API verze 7. Knihovna Design obsahuje další třídy, které se starají o vzhled a funkce Androidu 5.0+ na starších platformách až do API 14. Nejzajímavějším příkladem je kupříkladu pravé boční menu NavigationView. Knihovna RecyclerView obsahuje možnost vytvoření seznamu, který má velké množství nastavení a rovněž velkou míru přizpůsobení včetně animací. Knihovna CardView obsahuje menu ve formě karet, které lze použít pro tvorbu uživatelského rozhraní. A nakonec knihovna Preference obsahuje možnost tvorby menu s nastavením aplikace.

5.5.2 **Další knihovny**

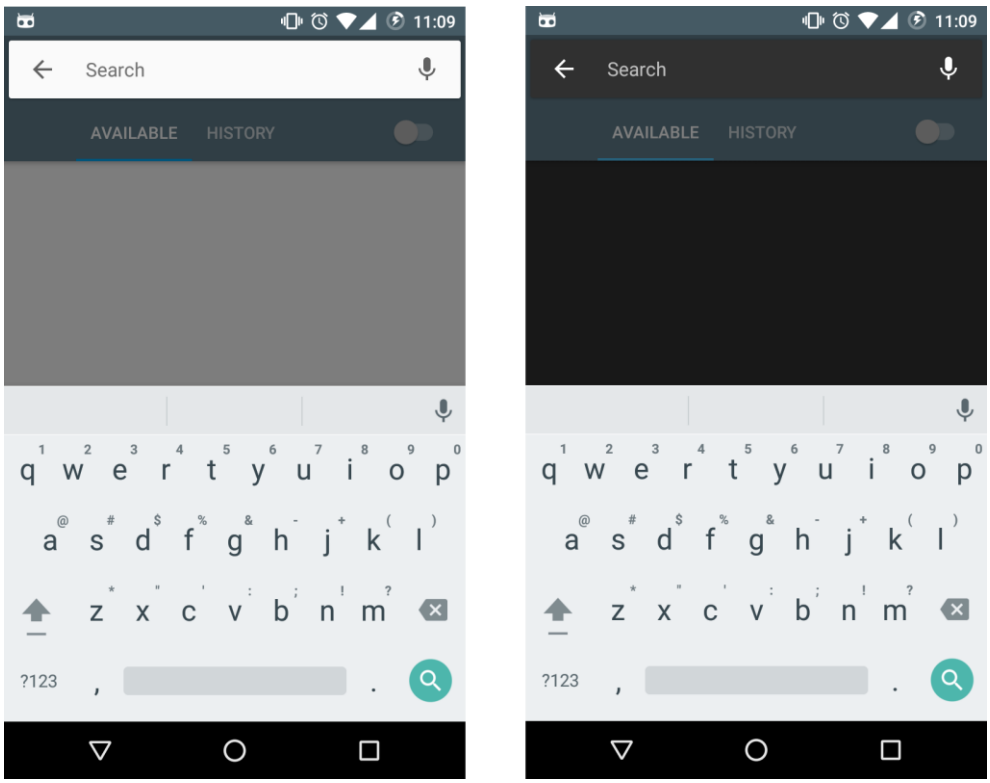
Google poskytuje velké množství knihoven, které usnadňují vývojáři programování funkcí aplikace, ale také i tvorbu uživatelského rozhraní. V průběhu vývoje aplikací a této diplomové práce jsem narazil na nutnost vytvoření dalších knihoven. Dvě z těchto knihoven jsem také implementoval do výsledné aplikace pro tuto diplomovou práci.

Aplikace obsahuje mou vlastní knihovnu s názvem Design. Obsahem této knihovny jsou velmi jednoduché třídy, které obsahují některé prvky pro uživatelské rozhraní. Například třída AboutView slouží k zobrazení informačních karet o knihovnách využitých v aplikaci. Třída MarkView se využívá pro zobrazení grafů ve StatisticsActivity, kde zobrazuje počet zpráv v databázových tabulkách CHAT_TABLE. Třída ErrorLayout slouží k zobrazení obrazovky s chybou při pokusu o práci s Bluetooth nebo Wi-Fi, aniž by byly tyto technologie na mobilním zařízení zapnuté. Třída FabBehavior se stará o posun plovoucího tlačítka při práci se seznamem zařízení a nakonec třída CheckableView slouží pro zobrazení kulaté barevné ikony u každé zprávy nebo nalezeného zařízení.

Knihovnu pro vyhledávání, která by splňovala specifikace dané Material Designem, Google neposkytuje a jediné místo, kde používá svoje neveřejné řešení, je obchod s aplikacemi Google Play. Aplikace tedy dále využívá mou vlastní knihovnu SearchView, která je implementována v aplikaci. Tato knihovna podporuje dvě verze vyhledávací lišty, pamatuje si historii vyhledávání, zobrazuje návrhy pro vyhledávání a rovněž obsahuje vzhled ve stylu Material Design, který lze upravovat. V aplikaci tato knihovna slouží k vyhledávání v zařízeních, ke kterým se aplikace připojila. Knihovna SearchView je veřejně dostupná na serveru GitHub [6] a v repozitářích jCenter na serveru Bintray [5]. Na obrázcích níže lze vidět vyhledávací okno knihovny.



Obrázek 1.17: *Grafy ve StatisticsActivity*



Obrázek 1.18: *Knihovna SearchView*

Závěr

Volba této diplomové práce mi zpětně přijde jako výborné rozhodnutí. Mobilními operačními systémy jsem se zabýval v několika předmětech ve studiu a operační systém Android je také mou velikou zálibou. Proto jsem velmi rád, že jsem díky této diplomovou prací získal možnost předvést své znalosti a také možnost vytvořit Android aplikaci, která je ukázkou možností aktuálního komunikačního API v operačním systému Android. Díky studiu komunikačních technologií Bluetooth a Wi-Fi Direct jsem získal množství nových znalostí, kterých doufám následně využiji v zaměstnání. Zároveň jsem dostal některé nápady na mé potencionální budoucí aplikace a knihovny.

Vzhledem k omezení technologie Bluetooth a Wi-Fi Direct a jejich aktuálních softwarových vývojářských nástrojů v Android SDK nebylo možné naprogramovat aplikaci, která bude plně využívat Ad-hoc způsobu komunikace prostřednictvím přímé komunikace zařízení tak, aby každé zařízení fungovalo jako samostatný uzel. Výsledná aplikace podporuje zabezpečenou a nezabezpečenou komunikaci prostřednictvím Bluetooth a Wi-Fi Direct. Tato komunikace probíhá pouze ve skupině, kde jedno ze zařízení tvoří server a zbytek jsou klienti. Tato komunikace pomocí zpráv, šifrovaných pomocí šifrování AES a ukládaná do implementované databáze, probíhá na pozadí, takže není nutné mít aplikaci stále zapnutou. Aplikace také umožňuje vyhledání Bluetooth zařízení a Wi-Fi Direct služby. Dále nabízí omezenou implementaci vlastního přenosového protokolu, který umožňuje komunikovat klientům v rámci skupiny pro Wi-Fi Direct nebo pro Bluetooth v rámci Piconetu. V aplikace lze rovněž nalézt vyhledávání v uložených klientech.

Bluetooth a Wi-Fi Direct patří k v Androidu k velmi omezeným API a na internetu existuje velmi málo českých článků a publikací, které by celkově popisovaly způsoby a omezení při tvorbě aplikací s podporou těchto technologií. Proto může tato diplomová práce sloužit jako cenný zdroj informaci pro vývoj aplikací s podporou klasického Bluetooth a služeb nad Wi-Fi Direct.

Velmi rád bych pokračoval ve vývoji této aplikace. Pro další vývoj bych doporučil úpravy rozhraní, práce se zprávami a úpravu a testování přenosového protokolu pro komunikaci mezi klienty přes server na více zařízeních.

Použitá literatura

[1] Android. Android [online]. [cit. 2016-03-21]. Dostupné z: https://www.android.com/

[2] Android TV. Android [online]. [cit. 2016-03-21]. Dostupné z: https://www.android.com/tv/

[3] Android Wear. Android [online]. [cit. 2016-03-21]. Dostupné z: https://www.android.com/wear/

[4] Kotlin Programming Language. Kotlin [online]. [cit. 2016-03-21]. Dostupné z: https://kotlinlang.org/

[5] SearchView. Bintray [online]. [cit. 2016-03-21]. Dostupné z: https://bintray.com/lapism/maven/searchview/view

[6] SearchView. GitHub [online]. [cit. 2016-03-21]. Dostupné z: https://github.com/lapism/SearchView

[7] Online UUID Generator Tool. UUID Generator [online]. [cit. 2016-03-21]. Dostupné z: https://www.uuidgenerator.net/

[8] Material design. Google design guidelines [online]. [cit. 2016-03-21]. Dostupné z: https://www.google.com/design/spec/material-design/introduction.html

[9] Xamarin. Microsoft Xamarin [online]. [cit. 2016-03-21]. Dostupné z: https://www.xamarin.com/

[10] Android NDK. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/ndk/index.html

[11] Android Studio + SDK. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/sdk/index.html

[12] Android ADT plugin. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/tools/sdk/eclipse-adt.html

[13] Android.bluetooth. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/reference/android/bluetooth/package-summary.html

[14] Android.net.wifi. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/reference/android/net/wifi/package-summary.html

[15] Android.net.wifi.p2p. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/reference/android/net/wifi/p2p/package-summary.html

[16] Android.net.wifi.p2p.nsd. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/reference/android/net/wifi/p2p/nsd/package-summary.html

[17] Android.bluetooth.le. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/reference/android/bluetooth/le/package-summary.html

[18] Android.net.nsd. Android [online]. [cit. 2016-03-21]. Dostupné z: http://developer.android.com/reference/android/net/nsd/package-summary.html

[19] Play Store. Google [online]. [cit. 2016-04-24]. Dostupné z: https://play.google.com/store

Seznam příloh

Součástí diplomové práce je příloha na DVD.

DVD obsahuje diplomovou aplikaci pro Android Studio. Zdrojové kódy aplikace jsou uloženy v souboru LAP036_DP_APP.zip.

DVD obsahuje instalační soubor aplikace ve formátu APK. Soubor je uložen pod názvem app.apk.

DVD obsahuje diplomovou práci ve formátu DOCX a PDF/A. Soubory jsou uloženy pod názvy LAP036_DP.docx a LAP036_DP.pdf.

DVD obsahuje schéma sítě Bluetooth Piconet ve formátu PNG. Soubor je uložen pod názvem bluetooth_piconet.png.

DVD obsahuje schéma sítě Bluetooth Scatternet ve formátu PNG. Soubor je uložen pod názvem bluetooth_scatternet.png.

DVD obsahuje schéma sítě Wi-Fi Direct Group ve formátu PNG. Soubor je uložen pod názvem wifi_direct_group.png.

DVD obsahuje třídní diagram aplikace ve formátu PNG. Soubor je uložen pod názvem class_diagram.png.

DVD obsahuje E-R diagram databáze aplikace ve formátu PNG. Soubor je uložen pod názvem er_diagram.png.